Techniques for Exploiting Unlabeled Data

Mugizi Robert Rwebangira

CMU-CS-08-164
October 2008

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Avrim Blum, CMU (Co-Chair)
John Lafferty, CMU (Co-Chair)
William Cohen, CMU
Xiaojin (Jerry) Zhu, Wisconsin

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under contract no. IIS-0427206, National Science Foundation under contract no. CCR-0122581, National Science Foundation under contract no. IIS-0312814, US Army Research Office under contract no. DAAD190210389, and SRI International under contract no. 03660211. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
OCT 2008		**00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Techniques for Exploiting Unlabeled Data	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**114**	

Keywords: semi-supervised, regression, unlabeled data, similarity

For my family

Abstract

In many machine learning application domains obtaining labeled data is expensive but obtaining unlabeled data is much cheaper. For this reason there has been growing interest in algorithms that are able to take advantage of unlabeled data. In this thesis we develop several methods for taking advantage of unlabeled data in classification and regression tasks.

Specific contributions include:

- A method for improving the performance of the graph mincut algorithm of Blum and Chawla [12] by taking randomized mincuts. We give theoretical motivation for this approach and we present empirical results showing that randomized mincut tends to outperform the original graph mincut algorithm, especially when the number of labeled examples is very small.

- An algorithm for semi-supervised regression based on manifold regularization using local linear estimators. This is the first extension of local linear regression to the semi-supervised setting. In this thesis we present experimental results on both synthetic and real data and show that this method tends to perform better than methods which only utilize the labeled data.

- An investigation of practical techniques for using the Winnow algorithm (which is not directly kernelizable) together with kernel functions and general similarity functions via unlabeled data. We expect such techniques to be particularly useful when we have a large feature space as well as additional similarity measures that we would like to use together with the original features. This method is also suited to situations where the best performing measure of similarity does not satisfy the properties of a kernel. We present some experiments on real and synthetic data to support this approach.

Acknowledgments

First I would like to thank my thesis committee members. Avrim Blum has been the most wonderful mentor and advisor any student could ask for. He is infinitely patient and unselfish and he always takes into account his students particular strengths and interests. John Lafferty is a rich fountain of ideas and mathematical insights. He was very patient with me as I learned a new field and always seemed to have the answers to any technical questions I had. William Cohen has a tremendous knowledge of practical machine learning applications. He helped me a lot by sharing his code, data and insights. I greatly enjoyed my collaboration with Jerry Zhu. He has an encyclopedic knowledge of the semi-supervised learning literature and endless ideas on how to pose and attack new problems.

I spent six years at Carnegie Mellon University. I thank the following collaborators, faculties, staffs, fellow students and friends, who made my graduate life a very memorable experience: Maria Florina Balcan, Paul Bennett,Sharon Burks,Shuchi Chawla, Catherine Copetas, Derek Dreyer, Christos Faloutsos, Rayid Ghani, Anna Goldenberg, Leonid Kontorovich,Guy Lebanon, Lillian Lee, Tom Mitchell, Andrew W Moore, Chris Paciorek, Francisco Pereira, Pradeep Ravikumar, Chuck Rosenberg, Steven Rudich, Alex Rudnicky, Jim Skees, Weng-Keen Wong,Shobha Venkataraman, T-H Hubert Chan, David Koes, Kaustuv Chaudhuri, Martin Zinkevich, Lie Gu, Steve Hanneke, Sumit Jha, Ciera Christopher, Hetunandan Kamichetty, Luis Von Ahn, Mukesh Agrawal, Jahanzeb Sherwani, Kate Larson, Urs Hengartner, Wendy Hu, Michael Tschantz, Pat Klahr, Ayorkor Mills-Tettey, Conniel Arnold, Karumuna Kaijage, Sarah Belousov, Varun Gupta, Thomas Latoza, Kemi Malauri, Randolph Scott-McLaughlin, Patricia Snyder, Vicky Rose Bhanji, Theresa Kaijage, Vyas Sekar, David McWherter, Bianca Schroeder, Bob Harper, Dale Houser, Doru Balcan, Elizabeth Crawford, Jason Reed, Andrew Gilpin, Rebecca Hutchinson, Lea Kissner, Donna Malayeri, Stephen Magill, Daniel Neill, Steven Osman, Peter Richter, Vladislav Shkapenyuk, Daniel Spoonhower, Christopher Twigg, Ryan Williams, Deepak Garg, Rob Simmons, Steven Okamoto, Keith Barnes, Jamie Main,, Marcie Smith, Marsey Jones, Deb Cavlovich, Amy Williams, Yoannis Koutis, Monica Rogati, Lucian Vlad-Llita, Radu Niculescu, Anton Likhodedov, Himanshu Jain, Anupam Gupta, Manuel Blum, Ricardo Silva, Pedro Calais, Guy Blelloch, Bruce Maggs, Andrew Moore, Arjit Singh, Jure Leskovec, Stano Funiak, Andreas Krause, Gaurav Veda, John Langford, R. Ravi, Peter Lee, Srinath Sridhar, Virginia Vassilevska, Jernej Barbic, Nikos Hardevallas . Of course there are many others not included here. My apologies if I left your name out. In particular, I thank you if you are reading this thesis.

Finally I thank my family. My parents Magdalena and Theophil always encouraged me to reach for the sky. My sisters Anita and Neema constantly asked me when I was going to graduate. I could not have done it without their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Summary

In the modern era two of the most significant trends affecting the use and storage of information are the following:

1. The rapidly increasing speed of electronic microprocessors.
2. The even more rapidly increasing capacity of electronic storage devices.

The latter has allowed the storage of vastly greater amounts of information than was possible in the past and the former has allowed the use of increasingly computationally intensive algorithms to process the collected information.

However, in the past few decades for many tasks the supply of information has outpaced our ability to effectively utilize it. For example in classification we are supplied with pairs of variables (X, Y) and we are asked to come up with a function that predicts the corresponding Y values for a new X. For example we might be given the images of handwritten digits and

the corresponding digit that they represent and be asked to learn an algorithm that automatically classifies new images into digits. Such an algorithm has many practical applications, for example the US Postal Service uses a similar algorithm for routing its mail[41].

The problem is that obtaining the initial training data which we need to use for learning can be expensive and might even require human intervention to label each example. In the previous example we would need someone to examine every digit in our dataset and determine its classification. In this case the work would not require highly skilled labor but it would still be impractical if we have hundreds of thousands of unlabeled examples.

Thus, a natural question is whether and how we can somehow make use of unlabeled examples to aid us in classification. This question has recently been studied with increasing intensity and several algorithms and theoretical insights have emerged.

First, as with all learning approaches we will need to make assumptions about the problem in order to develop algorithms. These assumptions typically involve the relationship between the unlabeled examples and the labels we want to predict. Examples of such assumptions include:

Large Margin Assumption - The data can be mapped into a space such that there exists a linear separator with a"large margin" that separates the (true) positive and (true) negative examples. "Large margin" has a technical definition but intuitively it simply means that the positive and negative examples are far apart. TSVM [45] is an example of an algorithm that uses this assumption.

Graph Partition - The data can be represented as a graph such that the (true) positive and (true) negative examples form two distinct components. The graph mincut algorithm [12] which we will discuss in chapter 3 makes this type of assumption.

Cluster Assumptions - Using the given distance metric the (true) positive examples and the (true) negative examples fall into two distinct clusters. This assumption is fairly general and in particular the two previous assumptions can be regarded as special cases of this assumption.

We will discuss these assumptions further in chapter 2 of this thesis.

Furthermore, if we make certain assumptions about the unlabeled data that turn out to not be true in practice not only will the unlabeled data not be helpful, it may actually cause our algorithm to perform worse than it would have if it ignored the unlabeled data. This is especially true if we do not have enough labeled data to perform cross validation. Thus unlabeled data should certainly be incorporated with care in any learning process.

From this discussion it is clear that there probably does not exist any universally applicable semi-supervised learning algorithm which performs better than all other algorithms on problems of interest. The usefulness of any semi-supervised learning algorithm will depend greatly on what kinds of assumptions it makes and whether such assumptions are met in practice. Thus we need to develop a variety of techniques for exploiting unlabeled data and practical experience in addition to theoretical guidance in choosing the best algorithm for specific situations.

In this thesis we develop three such methods for exploiting unlabeled data. First, we present the randomized graph mincut algorithm. This is an extension of the graph mincut algorithm for semi-supervised learning proposed by Blum and Chawla [12]. When the number of labeled examples is small the original graph mincut algorithm has a tendency to give severely unbalanced labelings (i.e. to label almost all examples as positive or to label almost all examples as negative). We show how randomization can be used to address this problem and we present experimental

data showing substantially improved performance over the original graph mincut algorithm.

Secondly, we present an algorithm for semi-supervised regression. The use of unlabeled data for regression has not been as well developed as it has been in classification. In semi-supervised regression we are presented with pairs of values (X, Y) where the Y values are real numbers and in addition we have a set of unlabeled examples X'. The task is to estimate the Y value for the unlabeled examples. We present Local Linear Semi-supervised Regression which is a very natural extension of both local Linear Regression and of the Gaussian Fields algorithm for semi-supervised learning of Zhu et al. [85]. We present some experimental results showing that this algorithm usually performs better than purely supervised methods.

Lastly, we examine the problem of learning with similarity functions. There has been a vast tradition of learning with kernel function in the machine learning community. However kernel functions have strict mathematical definitions (e.g. kernel functions have to be positive semi-definite). Recently there has been growing interest in techniques for learning even when the similarity function does not satisfy the mathematical properties of a kernel function. The connection with semi-supervised learning is that many of these techniques require more examples than the kernel based methods but the additional data can be unlabeled. That is, we can exploit the similarity between examples for learning even when we don't have the labels. Thus if we have large amounts of unlabeled data we would certainly be interested in using suitably defined similarity functions to improve our performance.

In the rest of this chapter we will describe the problems that this thesis addresses in greater detail. In chapter 2 we will discuss some of the related work that exists in the literature. In chapters 3,4 and 5 we will describe the actual results that were obtained in studying these problems.

1.2 Problem Description

1.2.1 Semi-supervised Classification

The classification problem is central in machine learning and statistics. In this problem we are given as input pairs of variables $(X_1, Y_1), ...(X_m, Y_m)$ where the X_i are objects of the type that we want to classify (for example documents or images) and the Y_i are the corresponding labels of the X_i (for example if the X_i are newspaper articles then the Y_i might indicate whether X_i is an article about machine learning). The goal is to minimize error rate on future examples X whose labels are not known. The special case where Y_i can only have two possible values is known as binary classification. This problem is also called supervised classification to contrast it with semi-supervised classification.

This problem has been extensively studied in the machine learning community and several algorithms have been proposed. A few of the algorithms which gained broader acceptance are perceptron, neural nets, decision trees and support vector machines. We will not discuss supervised classification further in this thesis but there are a number of textbooks which provide a good introduction to these methods [31, 32, 39, 56, 77, 78].

In the semi-supervised classification problem, in addition to labeled examples $(X_1, Y_1), ...(X_m, Y_m)$ we also receive unlabeled examples $X_{m+1}, ...X_n$. Thus we have m unlabeled examples and $n-m$ unlabeled examples.

In this setting we can define two distinct kinds of tasks:

1. To learn a function that takes any X and computes a corresponding Y.
2. To compute a corresponding Y_i for each of our unlabeled examples without necessarily

producing a function that makes a prediction for any new point X (this is sometimes referred to as *transductive classification*).

This problem only began to receive extensive attention in the early 90s although several algorithms were known before that. Some of the algorithms that have been proposed for this problem include the Expectation-Maximization algorithm proposed by Dempster, Laird and Rubin[30], the co-training algorithm proposed by Blum and Mitchell[13], the graph mincut algorithm proposed by Blum and Chawla[12], the Gaussian Fields algorithm proposed by Zhu, Gharamani and Lafferty[85] and Laplacian SVM proposed by Sindhwani, Belkin and Niyogi[68].We will discuss these algorithms further in chapter 2 of this thesis.

This area is still the subject of a very active research effort. A number of researchers have attempted to address the question of "Under what circumstances can unlabeled data be useful in classification" from a theoretical point of view [3, 22, 59, 80] and there also has been great interest from industrial practitioners who would like to make the best use of their unlabeled data.

1.2.2 Semi-supervised Regression

Regression is a fundamental tool in statistical analysis. At its core regression aims to model the relationship between 2 or more random variable. For example, an economist might want to investigate whether more education leads to an increased income. A natural way to accomplish this is to take number of years of education as the dependent variable and annual income as the independent variable and to use regression analysis to determine their relationship.

Formally, we are given as input $(X_1, Y_1), ...(X_n, Y_n)$ where the X_i are the dependent variables and Y_i are the independent variables. We want to predict for any X the value of the correspond-

ing Y. There are two main types of techniques used to accomplish this:

1. Parametric regression: In this case we assume that the relationship between the variable is of a certain type (e.g. a linear relationship) and we are concerned with learning the parameters for a relationship of that type which best fit the data.
2. Non-parametric regression: In this case we do not make any assumptions about the type of relationship that holds between the variables, but we derive this relationship directly from the data.

Regression analysis is heavily used in the natural sciences and in social sciences such as economics, sociology and political science. A wide variety of regression algorithms are used including linear regression, polynomial regression and logistic regression among the parametric methods and kernel regression and local linear regression among the non-parametric methods. A further discussion of such methods can be found in any introductory statistics textbook[77, 78].

In semi-supervised regression in addition to getting the dependent and independent variables X and Y we are also given an addition variable R which indicates whether or not we observe that value of Y. In other words we get data $(X_1, Y_1, R_1), ...(X_n, Y_n, R_n)$ and we observe Y_i only if $R_i = 1$.

We note that the problem of semi-supervised regression is more general than the semi-supervised classification problem. In the latter case the Y_i are constrained to have only a finite number of possible values whereas in regression the Y_i are assumed to be continuous. Hence some algorithms designed for semi-supervised classification (e.g. graph mincut[12]) are not applicable to the more general semi-supervised regression problem. Other algorithms such as Gaussian Fields [85]) are applicable to both problems.

Although semi-supervised regression has received less attention than semi-supervised classification a number of methods have been developed dealing specifically with this problem. These include the transductive regression algorithm proposed by Cortes and Mohri [23]and co-training style algorithms proposed by Zhou and Li [82], Sindhwani et al.[68] and Brefeld et al.[17]. We will discuss these related approaches in more detail in chapter 2.

1.2.3 Learning with Similarity functions

In many machine learning algorithms it is often useful to compute a measure of the similarity between two objects. Kernel functions are a popular way of doing this. A kernel function $K(x, y)$ takes two objects and outputs a positive number that is a measure of the similarity of the objects. A kernel function must also satisfy the mathematical condition of positive semi-definiteness.

It follows from Mercer's theorem that any kernel function can also be interpreted as an inner product in some Euclidean vector space. This allows us to take advantage of the representation power of high-dimensional vector spaces without having to explicitly represent our data in such spaces. Due to this insight (known as the "kernel trick") kernel methods have become extremely popular in the machine learning community, resulting in widely used algorithms such as Support Vector Machines [62, 64, 65].

However, sometimes it turns out that a useful measure of similarity does not satisfy the mathematical conditions to be a kernel function. An example is the Smith-Waterman score [76] which is a measure of the alignment of two protein sequences. It is widely used in molecular biology and has been empirically successful in predicting the similarity of proteins. However, the Smith-Waterman score is not a valid kernel function and hence cannot be directly used in algorithms

such as Support Vector Machines.

Thus there has been a growing interest in developing more general methods for utilizing similarity functions which may not satisfy the positive semi-definite requirement. Recently, Balcan and Blum [2] proposed a general theory of learning with similarity functions. They give a natural definition of similarity function which contains kernel functions as a sub-class and show that effective learning can be done in this framework.

Although the work in this area is still very preliminary, there is strong interest due to the practical benefits of being able to exploit more general kinds of similarity functions.

Chapter 2

Related Work

In the last decade or so there has been a substantial amount of work on exploiting unlabeled data. The survey by Zhu [83] is the most up to date summary of work in this area. The recent book edited by Chapelle et al. [25] is a good summary of work done up to 2005 and attempts to synthesize the main insights that have been gained. In this chapter we will mainly highlight the work that is most closely related to our own.

2.1 Semi-supervised Classification

As with supervised classification, semi-supervised classification methods fall into two categories:

1. Generative methods which attempt to model the statistical distribution that generates the data before making predictions.
2. Discriminative methods which directly make predictions without assumptions about the distribution the data comes from.

2.1.1 Generative Methods

Expectation-Maximization (EM)

Suppose the examples are $(x_1, x_2, x_3, ..., x_n)$ and the labels are $(y_1, y_2, y_3, ...y_l)$. Generative models try to model the class conditional densities $p(x|y)$ and then apply Bayes' rule to compute predictive densities $p(y|x)$.

BAYES' RULE: $p(y|x) = \frac{p(x|y)p(y)}{\int_y p(x|y)p(y)dy}$

In this setting unlabeled data gives us more information about $p(x)$ and we would like to use this information to improve our estimate of $p(y|x)$. If information about $p(x)$ is not useful to estimating $p(y|x)$ (or we do not use it properly) then unlabeled data will not help to improve our predictions.

In particular if our assumptions about the distribution the data comes from are incorrect then unlabeled data can actually degrade our classification accuracy. Cozman and Cohen explore this effect in detail for generative semi-supervised learning models [27].

However, if our generative model for x is correct then any additional information about $p(x)$ will generally be useful. For example suppose that $p(x|y)$ is a gaussian. Then the task becomes to estimate the parameters of the gaussian. This can easily be done with sufficient labeled data, but if there a few labeled examples unlabeled data can greatly improve the estimate. Castelli and Cover [22] provide a theoretical analysis of this scenario.

A popular algorithm to use in the above scenario is the Expectation-Maximization algorithm proposed by Dempster, Laird and Rubin [30]. EM is an iterative algorithm that can be used to

compute Maximum-Likelihood estimates of parameters when some of the relevant information is missing. It is guaranteed to converge and is fairly fast in practice. However it is only guaranteed to converge to a local minima.

An advantage of generative approaches is that knowledge about the structure of a problem can be naturally incorporated by modelling it in the generative model. The work by Nigam et al. [57] on modelling text data is an example of this approach. They model text data using a Naïve Bayes classifer and then estimate the parameters using EM. But as stated before, if the assumptions are incorrect then using unlabeled data can lead to worse inferences than completely ignoring the unlabeled data.

2.1.2 Discriminative Methods

As stated before, in semi-supervised classification we wish to exploit the relationship between $p(x)$ and $p(y|x)$. with generative methods we assume that the distribution generating the data has a certain form and the unlabeled data helps us to learn the parameters of the distribution more accurately. Discriminative methods do not bother to try and model the distribution generating the data $p(x)$, hence in order to use unlabeled data we have to make "a priori" assumptions on the relationship between $p(x)$ and $p(y|x)$.

We can categorize discriminative methods for semi-supervised learning by the kind of assumption they make on relationship between the distribution of examples and the conditional distribution of the labels. In this survey we will focus on a family of algorithms that use what is known as the Cluster Assumption.

The Cluster Assumption

The cluster assumption posits a simple relationship between $p(x)$ and $p(y|x)$: $p(y|x)$ should change slowly in regions where $p(x)$ has high density. Informally this is equivalent to saying that examples that are "close" to each other should have "similar" labels.

Hence, gaining information about $p(x)$ gives information about the high density region (clusters) in which examples should be given the same label. This can also be viewed as a "reordering" of the hypothesis space: We give preference to those hypotheses which do not violate the cluster assumption.

The cluster assumption can be realized in variety of ways and leads to a number of different algorithms. Most prominent are **graph based methods** in which a graph is constructed that contains all the labeled and unlabeled examples as nodes and the weight of an edge between nodes indicates the similarity of the corresponding examples.

We note that graph based methods are inherently transductive although it is easy to extend them to the inductive case by taking the predictions of the semi-supervised classification as training data and then using a non-parametric supervised classification algorithm such as k-nearest neighbor on the new example. In this case classification time will be significantly faster than training time. Another option is to rerun the entire algorithm when presented with a new example.

Graph Based Methods

Graph Mincut

The graph mincut algorithm proposed by Blum and Chawla [12] was one of the earliest graph based methods to appear in the literature. The essential idea of this algorithm is to convert semi-supervised classification into an *s*-*t* mincut problem[26] . The algorithm is as follows:

1. Construct a graph joining examples which have similar labels. The edges may be weighted or unweighted.

2. Connect the positively labeled examples to a “source” node with “high-weight” edges. Connect the negatively labeled examples to a “sink” node with “high-weight” edges.

3. Obtain an *s*-*t* minimum cut of the graph and classify all the nodes connected to the “source” as positive examples and all the nodes connected to the sink as negative examples.

There is considerable freedom in the construction of the graph. Properties that are empirically found to be desirable are that the graph should be connected, but that it shouldn’t be too dense.

As stated the algorithm only applies to binary classification, but one can easily imagine extending it to general classification by taking a multi-way cut. However, this would entail a significant increase in the running time as multi-way cut is known to be NP-complete. We note that there has been substantial work in the image-segmentation literature on using multi-way cuts (e.g. Boykov et al. [16].)

This algorithm is attractive because it is simple to understand and easy to implement. However, a significant problem is that it can often return very "unbalanced" cuts. More precisely, if the number of labeled examples is small, the *s-t* minimum cut may chop off a very small piece of the graph and return this is as the solution. Furthermore, there is no known natural way to "demand" balanced cut without running into a significantly harder computational problem. (The Spectral Graph Transduction algorithm proposed by Joachims is one attempt to do this efficiently) [44].

In chapter 3 of this thesis we report on techniques for overcoming this problem of unbalanced cuts and show significant improvement in results compared to the original graph mincut algorithm.

Gaussian Fields Method

This central idea of this algorithm is to reduce semi-supervised classification to finding the minimum energy of a certain Gaussian Random Field[85].

The algorithm is as follows:

1. Compute a weight W_{ij} between all pairs of examples.
2. Find real values f that minimize the energy functional $E(f) = \frac{1}{2}\sum_{i,j} w_{ij}(f(i) - f(j))^2$ (where the value of the labeled f_i's are fixed.)
3. Assign each (real) f_i to one of the discrete labels.

It turns out that the solution to step (2) is closely connected to random walks, electrical networks and spectral graph theory. For example, if we think of each edge as a resistor, it is

equivalent to placing a battery with positive terminal connected to the positive labeled examples, negative terminal connected to the negative labeled examples and measuring voltages at the unlabeled points.

Further, this method can also be viewed as a continuous relaxation of the graph mincut method. More importantly the solution can be computed using only matrix operations for the cost of inverting a $u \times u$ matrix (where u is the number of labeled examples).

A major advantage of this method is that is fairly straightforward to implement. In addition, unlike graph-mincut it can be generalized to the multi-label case without a significant increase in complexity.

However, note that as stated it could still suffer from an "unbalanced" labelings. Zhu et al. [85] report on using a "Class Mass Normalization" heuristic to force the unlabeled examples to have the same class proportions as the labeled examples.

Laplacian SVM

The main idea in this method is to take the SVM objective function and add an extra regularization penalty that penalizes similar examples that have different labels[6].

More precisely the solution for SVM can be stated as

$f^* = argmin f \in \mathcal{H}_\mathcal{K} = \frac{1}{l} \sum_{i=1}^{l} V(x_i, y_i, f) + \gamma ||f||_K^2$

$V(x_i, y_i, f)$ is some loss function such as squared loss $(y_i - f(x_i))^2$ or soft margin loss

$max\{0, 1 - y_i f(x_i)\}$.

The second term is a regularization that imposes smoothness condition on possible solutions.

In Laplacian SVM the solution is the following:

$$f^* = argmin f \in \mathcal{H}_\mathcal{K} = \frac{1}{l}\sum_{i=1}^{l} V(x_i, y_i, f) + \gamma_A ||f||_K^2 + \frac{\gamma_I}{(u+l)^2}\sum_{i,j} w_{ij}(f(i) - f(j))^2$$

The extra regularization term imposes an additional smoothness condition over both the labeled and unlabeled data. We note that the extra regularization term is the same as the objective function in the Gaussian Fields method.

An advantage of this method is that it easily extends to the inductive case since the representer theorem still applies.

$f^*(x) = \sum_{i=1}^{l+u} \alpha_i K(x_i, x)$. (Representer Theorem)

On the other hand, implementation of this method will necessarily be more complicated since it involves solving a quadratic program.

Transductive SVM (TSVM)

Transductive SVM was first proposed by Vapnik [[74],[75]] as a natural extension of the SVM algorithm to unlabeled data. The idea is very simple: instead of simply finding a hyperplane that maximizes the margin on the labeled examples we want to find a hyperplane that maximizes the margin on the labeled and unlabeled examples.

Unfortunately this simple extension fundamentally changes the nature of the optimization problem. In particular the original SVM leads to a convex optimization problem which has a unique solution and can be solved efficiently. TSVM does not lead to a convex optimization problem (due to the non-linear constraints imposed by the unlabeled data) and in particular there is no known method to solve it efficiently.

However there are encouraging empirical results: Thorsten Joachims proposed an algorithm that works by first obtaining a supervised SVM solution then doing a coordinate descent to improve the objective function. It showed significant improvement over purely supervised methods and scaled up to 100,000 examples [45].

Die Bie and Cristianini proposed a semi-definite programming relaxation of the TSVM[9][10]. They then further proposed an approximation of this relaxation and showed that at least in some cases they attain better performance than the version of TSVM proposed by Joachims. However, their method is not able to scale to more than 1000 examples.

TSVM is very attractive from a theoretical perspective as it inherits most of the justifications of SVM and the large-margin approach, however as of now it is still very challenging from an implementation point of view.

Spectral Graph Transducer (SGT)

The central idea in this method is to ensure that the proportions of positive and negative examples is the same in the labeled and unlabeled sets[44]. As we noted this is an issue in several methods based on the Cluster Assumption such as graph mincut[12] and Gaussian Fields[85] .

SGT addresses this issue by reducing the problem to an unconstrained ratio-cut in a graph with additional constraints to take into account the labeled data. Since the resulting problem is still NP-Hard, the real relaxation of this problem is used for the actual solution. It turns out that this can be solved efficiently.

Joachims reports some impressive experimental results in comparison with TSVM, kNN and SVM. Furthermore, the algorithm has an easy to understand and attractive intuition. However, the reality is that it is not (so far) possible to solve the original problem optimally and it is not clear what the quality of the solution of the relaxation is. Some experimental results by Blum et al. [14] indicate that SGT is very sensitive to fine tuning of its parameters.

2.2 Semi-supervised Regression

While some semi-supervised classification such as Gaussian Fields can be directly applied to semi-supervised regression without any modifications, there has been relatively little work directly targeting semi-supervised regression. It is not clear if this is due to lack of interest from researchers or lack of demand from practitioners.

Correspondingly the theoretical insights and intuitions are not as well developed for semi-supervised regression as they are for semi-supervised classification. In particular so far no taxonomy for semi-supervised regression methods has been proposed that corresponds to the taxonomy for semi-supervised classification methods.

Still, the overall task remains the same: to use information about the distribution of x to improve our estimates of $p(y|x)$. To this end, we need to make an appropriately useful assumption.

The Manifold assumption is one such candidate: We assume that the data lies along a low dimensional manifold.

An example of this would be if the true function we are trying to estimate is a 3 dimensional spiral. If we only have a few labeled examples, it would be hard to determine the structure of the entire manifold. However, if we have sufficiently large amount of unlabeled data, the manifold becomes much easier to determine. The Manifold assumption also implies the Smoothness assumption: Examples which are close to each other, have similar labels. In other words we expect the function to not "jump" suddenly.

2.2.1 Transductive Regression algorithm of Cortes and Mohri

The transductive regression algorithm of Cortes and Mohri[23] minimizes an objective function of the following form:

$$\mathcal{E}(h) = ||w||^2 + C\sum_{i=1}^{m}(h(x_i) - y_i)^2 + C'\sum_{i=m+1}^{m+u}(h(x_i) - \hat{y}_i)^2$$

Here the hypothesis h is a linear function of the form $h(x) = w \cdot \Phi(x)$ where w is a vector in vector space $\mathcal{F}$, x is an element in $\mathcal{X}$ and Φ is a feature map from $\mathcal{X}$ to $\mathcal{F}$. In addition C and C' are regularization parameters, y_i is the values of the i^{th} labeled example and $\hat{y}_i$ is an estimate for the i^{th} unlabeled example. The task is to estimate the vector w of weights.

A strong attraction of this method is that it is quite computationally efficient: It essentially only requires the inversion of an $D \times D$ matrix where D is the dimension of $\mathcal{F}$ the space into which the examples are mapped. Cortes and Mohri [23] report impressive empirical results on various regression datasets. However, it is not clear how much performance is sacrificed by as-

suming the hypothesis is of the form $h(x) = w \cdot \Phi(x)$. It would be interesting to compare with a purely non-parametric method.

2.2.2 COREG (Zhou and Li)

The key idea of this method is to apply co-training to the semi-supervised regression task [82]. The algorithm uses two k-nearest neighbor regressors with different distance metrics, each of which labels the unlabeled data for the other regressor. The labeling confidence is estimated through consulting the influence of the labeling of unlabeled examples on the labeled ones.

Zhou and Li report positive experimental results on mostly synthetic datasets. However, since this was the first paper to deal with semi-supervised regression, they were not able to compare with more recent techniques.

The concept of applying co-training to regression is attractive, however it is not clear what unlabeled data assumptions are being exploited.

2.2.3 Co-regularization (Sindhwani et al.)

This technique also uses co-training but in a regularization framework[68].

More precisely the aim is to learn two different classifiers by optimizing an objective function that simultaneously minimizes their training error and their disagreement with each other.

Brefeld et al. [17] use the same idea, they are also propose a fast approximation that scales linearly in the number of training examples.

2.3 Learning with Similarity functions

In spite of (or maybe because of) the wide popularity of kernel based learning methods in the past decade, there has been relatively little work on learning with similarity functions that are not kernels.

The paper by Balcan and Blum [2] was the first to rigorously analyze this framework. The main contribution was to define a notion of good similarity function for learning which was intuitive and included the usual notion of a large margin kernel function.

Recently Srebro [70] gave an improved analysis with tighter bounds on the relation between large margin kernel functions and good similarity functions in the Balcan-Blum sense. In particular he showed that large margin kernel functions remain good when used with a hinge loss. He also gives examples where using a kernel function as a similarity function can produce worse margins although his results do not imply that the margin will always be worse if a kernel is used as a similarity function.

A major practical motivation for this line of research is the existence of domain specific similarity functions (typically defined by domain experts) which are known to be successful but which do not satisfy the definition of a kernel function. The Smith-Waterman score used in biology is a typical example of such a similarity function. The Smith-Waterman score is a measure of local alignment between protein sequences. It is considered by biologist to be the best measure of homology (similarity) between two protein sequences.

However, it does not satisfy the definition of a kernel function and hence cannot be used in

kernel based methods. To deal with this issue Vert et al. [76] construct a convolution kernel to "mimic" the behavior of the Smith-Waterman score. Vert et al. report promising experimental results, however it is highly likely that the original Smith-Waterman score provides a better measure of similarity than the kernelized version. Hence techniques that use the original similarity measure might potentially have superior performance.

In this work we will focus on using similarity functions with online algorithms like Winnow. One motivation for this is in our approach similarity functions are most useful if we have large amounts of unlabeled data. For large datasets we need fast algorithms such as Winnow.

Another advantage of Winnow specifically is that it can learn well even in presence of irrelevant features. This is important in learning with similarities as we will typically create several new features for each examples and only a few of the generated features may be relevant to the classification task.

Winnow was proposed by Littlestone [53] who analyzed some of its basic properties. Blum [11] and Dagan [28] demonstrated that Winnow could be used in real world tasks such as calendar scheduling and text classification. More recent experimental work has shown that versions of Winnow can be competitive with the best offline classifiers such as SVM and Logistic Regression [Bekkerman [5], Cohen & Carvalho [20] [21].]

Chapter 3

Randomized Mincuts

In this chapter we will describe the randomized mincut algorithm for semi-supervised classification. We will give some background and motivation for this approach and also give some experimental results.

3.1 Introduction

If one believes that "similar examples ought to have similar labels," then a natural approach to using unlabeled data is to combine nearest-neighbor prediction—predict a given test example based on its nearest labeled example—with some sort of self-consistency criteria, e.g., that similar *unlabeled* examples should, in general, be given the same classification. The graph mincut approach of Blum and Chawla [12] is a natural way of realizing this intuition in a transductive learning algorithm. Specifically, the idea of this algorithm is to build a graph on all the data (labeled and unlabeled) with edges between examples that are sufficiently similar, and then to partition the graph into a positive set and a negative set in a way that (a) agrees with the labeled data, and (b) cuts as few edges as possible. (An edge is "cut" if its endpoints are on different sides of the partition.)

The graph mincut approach has a number of attractive properties. It can be found in polynomial time using network flow; it can be viewed as giving the most probable configuration of labels in the associated Markov Random Field (see Section 3.2); and, it can also be motivated from sample-complexity considerations, as we discuss further in Section 3.4.

However, it also suffers from several drawbacks. First, from a practical perspective, a graph may have many minimum cuts and the mincut algorithm produces just one, typically the "leftmost" one using standard network flow algorithms. For instance, a line of n vertices between two labeled points s and t has $n-1$ cuts of size 1, and the leftmost cut will be especially unbalanced. Second, from an MRF perspective, the mincut approach produces the most probable joint labeling (the MAP hypothesis), but we really would rather label nodes based on their *per-node* probabilities (the Bayes-optimal prediction). Finally, from a sample-complexity perspective, if we could average over many small cuts, we could improve our confidence via PAC-Bayes style arguments.

Randomized mincut provides a simple method for addressing a number of these drawbacks. Specifically, we repeatedly add artificial random noise to the edge weights,[1] solve for the minimum cut in the resulting graphs, and finally output a fractional label for each example corresponding to the fraction of the time it was on one side or the other in this experiment. This is not the same as sampling directly from the MRF distribution, and is also not the same as picking truly random minimum cuts in the original graph, but those problems appear to be much more difficult computationally on general graphs (see Section 3.2).

A nice property of the randomized mincut approach is that it easily leads to a measure of con-

[1]We add noise only to existing edges and do not introduce new edges in this procedure.

fidence on the predictions; this is lacking in the deterministic mincut algorithm, which produces a single partition of the data. The confidences allow us to compute accuracy-coverage curves, and we see that on many datasets the randomized mincut algorithm exhibits good accuracy-coverage performance.

We also discuss design criteria for constructing graphs likely to be amenable to our algorithm. Note that some graphs simply do not have small cuts that match any low-error solution; in such graphs, the mincut approach will likely fail even with randomization. However, constructing the graph in a way that is very conservative in producing edges can alleviate many of these problems. For instance, we find that a very simple minimum spanning tree graph does quite well across a range of datasets.

PAC-Bayes sample complexity analysis [54] suggests that when the graph has many small cuts consistent with the labeling, randomization should improve generalization performance. This analysis is supported in experiments with datasets such as handwritten digit recognition, where the algorithm results in a highly accurate classifier. In cases where the graph does not have small cuts for a given classification problem, the theory also suggests, and our experiments confirm, that randomization may not help. We present experiments on several different datasets that indicate both the strengths and weaknesses of randomized mincuts, and also how this approach compares with the semi-supervised learning schemes of Zhu et al. [85] and Joacchims [44]. For the case of MST-graphs, in which the Markov random field probabilities *can* be efficiently calculated exactly, we compare to that method as well.

In the following sections we will give some background on Markov random fields, describe our algorithm more precisely as well as our design criteria for graph construction, provide sample-complexity analysis motivating some of our design decisions, and finally give some ex-

perimental results.

3.2 Background and Motivation

Markov random field models originated in statistical physics, and have been extensively used in image processing. In the context of machine learning, what we can do is create a graph with a node for each example, and with edges between examples that are similar to each other. A natural energy function to consider is

$$E(f) = \frac{1}{2}\sum_{i,j} w_{ij}|f(i) - f(j)| = \frac{1}{4}\sum_{i,j} w_{ij}(f(i) - f(j))^2$$

where $f(i) \in \{-1, +1\}$ are binary labels and w_{ij} is the weight on edge (i, j), which is a measure of the similarity between the examples. To assign a probability distribution to labelings of the graph, we form a random field

$$p_\beta(f) = \frac{1}{Z}\exp\left(-\beta E(f)\right)$$

where the partition function Z normalizes over all labelings. Solving for the lowest energy configuration in this Markov random field will produce a partition of the entire (labeled and unlabeled) dataset that maximally optimizes self-consistency, subject to the constraint that the configuration must agree with the labeled data.

As noticed over a decade ago in the vision literature [37], this is equivalent to solving for a minimum cut in the graph, which can be done via a number of standard algorithms. Blum and Chawla [12] introduced this approach to machine learning, carried out experiments on several datasets, and explored generative models that support this notion of self-consistency.

The minimum cut corresponds, in essence, to the MAP hypothesis in this MRF model. To produce Bayes-optimal predictions, however, we would like instead to sample directly from the

MRF distribution.

Unfortunately, that problem appears to be much more difficult computationally on general graphs. Specifically, while random labelings can be efficiently sampled *before* any labels are observed, using the well-known Jerrum-Sinclair procedure for the Ising model [42], after we observe the labels on some examples, there is no known efficient algorithm for sampling from the conditional probability distribution; see Dyer et al. [33] for a discussion of related combinatorial problems.

This leads to two approaches:

1. Try to approximate this procedure by adding random noise into the graph.
2. Make sure the graph is a tree, for which the MRF probabilities can be calculated exactly using dynamic programming.

Here, we will consider both.

3.3 Randomized Mincuts

The randomized mincut procedure we consider is the following. Given a graph G constructed from the dataset, we produce a collection of cuts by repeatedly adding random noise to the edge weights and then solving for the minimum cut in the perturbed graph.

In addition, now that we have a collection of cuts, we remove those that are highly unbalanced. This step is justified using a simple ϵ-cover argument (see Section 3.4), and in our experiments, any cut with less than 5% of the vertices on one side is considered unbalanced.[2]

[2]With only a small set of labeled data, one cannot in general be confident that the true class probabilities are

Finally, we predict based on a majority vote over the remaining cuts in our sample, outputting a confidence based on the margin of the vote. We call this algorithm "Randomized mincut with sanity check" since we use randomization to produce a distribution over cuts, and then throw out the ones that are obviously far from the true target function.

In many cases this randomization can overcome some of the limitations of the plain mincut algorithm. Consider a graph which simply consists of a line, with a positively labeled node at one end and a negatively labeled node at the other end with the rest being unlabeled. Plain mincut may choose from any of a number of cuts, and in fact the cut produced by running network flow will be either the leftmost or rightmost one depending on how it is implemented. Our algorithm will take a vote among all the mincuts and thus we will end up using the middle of the line as a decision boundary, with confidence that increases linearly out to the endpoints.

It is interesting to consider for which graphs our algorithm produces a true uniform distribution over minimum cuts and for which it does not. To think about this, it is helpful to imagine we collapse all labeled positive examples into a single node s and we collapse all labeled negative examples into a single node t. We can now make a few simple observations. First, a class of graphs for which our algorithm *does* produce a true uniform distribution are those for which all the s-t minimum cuts are disjoint, such as the case of the line above. Furthermore, if the graph can be decomposed into several such graphs running in parallel between s and t ("generalized theta graphs" [19]), then we get a true uniform distribution as well. That is because any minimum s-t cut must look like a tuple of minimum cuts, one from each graph, and the randomized mincut algorithm will end up choosing at random from each one.

On the other hand, if the graph has the property that some minimum cuts overlap with many

close to the observed fractions in the training data, but one *can* be confident that they are not extremely biased one way or the other.

others and some do not, then the distribution may not be uniform. For example, Figure 3.1 shows a case in which the randomized procedure gives a much higher weight to one of the cuts than it should ($\geq 1/6$ rather than $1/n$).

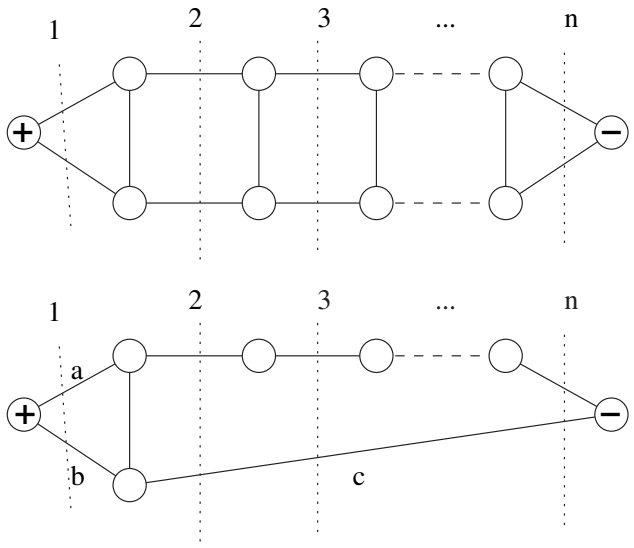

Figure 3.1: A case where randomization will not uniformly pick a cut

Looking at Figure 3.1 ,in the top graph, each of the n cuts of size 2 has probability $1/n$ of being minimum when random noise is added to the edge lengths. However, in the bottom graph this is not the case. In particular, there is a constant probability that the noise added to edge c exceeds that added to a and b combined (if a, b, c are picked at random from $[0, 1]$ then $\Pr(c > a + b) = 1/6$). This results in the algorithm producing cut $\{a, b\}$ no matter what is added to the other edges. Thus, $\{a, b\}$ has a much higher than $1/n$ probability of being produced.

3.4 Sample complexity analysis

3.4.1 The basic mincut approach

From a sample-complexity perspective, we have a transductive learning problem, or (roughly) equivalently, a problem of learning from a known distribution. Let us model the learning scenario as one in which first the graph G is constructed from data without any labels (as is done in our experiments) and then a few examples at random are labeled. Our goal is to perform well on the rest of the points. This means we can view our setting as a standard PAC-learning problem over the uniform distribution on the vertices of the graph. We can now think of the mincut algorithm as motivated by standard Occam bounds: if we describe a hypothesis by listing the edges cut using $O(\log n)$ bits each, then a cut of size k can be described in $O(k \log n)$ bits.[3] This means we need only $O(k \log n)$ labeled examples to be confident in a consistent cut of k edges (ignoring dependence on ϵ and δ).

In fact, we can push this bound further: Kleinberg [48], studying the problem of detecting failures in networks, shows that the VC-dimension of the class of cuts of size k is $O(k)$. Thus, only $O(k)$ labeled examples are needed to be confident in a consistent cut of k edges. kleinberg et al. [50] reduce this further to $O(k/\lambda)$ where λ is the size of the *global* minimum cut in the graph (the minimum number of edges that must be removed in order to separate the graph into two nonempty pieces, without the requirement that the labeled data be partitioned correctly).

One implication of this analysis is that if we imagine data as being labeled for us one at a time, we can plot the size of the minimum cut found (which can only increase as we see more labeled data) and compare it to the global minimum cut in the graph. If this ratio grows slowly with the number of labeled examples, then we can be confident in the mincut predictions.

[3]This also assumes the graph is connected — otherwise, a hypothesis is not uniquely described by the edges cut.

3.4.2 Randomized mincut with "sanity check"

As pointed out by Joachims [44], minimum cuts can at times be very unbalanced. From a sample-complexity perspective we can interpret this as a situation in which the cut produced is simply not small enough for the above bounds to apply given the number of labeled examples available. From this point of view, we can think of our mincut extension as being motivated by two lines of research on ways of achieving rules of higher confidence.

The first of these are PAC-Bayes bounds [52, 54]. The idea here is that even if no single consistent hypothesis is small enough to inspire confidence, if many of them are "pretty small" (that is, they together have a large prior if we convert our description language into a probability distribution) then we can get a better confidence bound by randomizing over them.

Even though our algorithm does not necessarily produce a true uniform distribution over all consistent minimum cuts, our goal is simply to produce as wide a distribution as we can to take as much advantage of this as possible. Furthermore results of Freund et al.[35] show that if we weight the rules appropriately, then we can expect a lower error rate on examples for which their vote is highly biased.

Again, while our procedure is at best only an approximation to their weighting scheme, this motivates our use of the bias of the vote in producing accuracy/coverage curves. We should also note that recently Hanneke[38] has carried out a much more detailed version of the analysis that we have only hinted at here.

The second line of research motivating aspects of our algorithm is work on bounds based on ϵ-cover size, e.g., Benedek et al.[7]. The idea here is that suppose we have a known distribution D and we identify some hypothesis h that has many similar hypotheses in our class with respect

to D. Then if h has a high error rate over a labeled sample, it is likely that *all* of these similar hypotheses have a high true error rate, *even if some happen to be consistent with the labeled sample*.

In our case, two specific hypotheses we can easily identify of this form are the "all positive" and "all negative" rules. If our labeled sample is even reasonably close to balanced — e.g., 3 positive examples out of 10 — then we can confidently conclude that these two hypotheses have a high error rate, and throw out *all highly unbalanced cuts*, even if they happen to be consistent with the labeled data.

For instance, the cut that simply separates the three positive examples from the rest of the graph is consistent with the data, but can be ruled out by this method.

This analysis then motivates the second part of our algorithm in which we discard all highly unbalanced cuts found before taking majority vote. The important issue here is that we can confidently do this even if we have only a very small labeled sample. Of course, it is possible that by doing so, our algorithm is never able to find a cut it is willing to use. In that case our algorithm halts with failure, concluding that the dataset is not one that is a good fit to the biases of our algorithm. In that case, perhaps a different approach such as the methods of Joachims[44] or Zhu et al. [85] or a different graph construction procedure is needed.

3.5 Graph design criteria

For a given distance metric, there are a number of ways of constructing a graph. In this section, we briefly discuss design principles for producing graphs amenable to the graph mincut algorithm. These then motivate the graph construction methods we use in our experiments.

First of all, the graph produced should either be connected or at least have the property that a small number of connected components cover nearly all the examples. If t components are needed to cover a $1 - \epsilon$ fraction of the points, then clearly any graph-based method will need t labeled examples to do well.[4]

Secondly, for a mincut-based approach we would like a graph that at least has some small balanced cuts. While these may or may not correspond to cuts consistent with the labeled data, we at least do not want to be dead in the water at the start. This suggests conservative methods that only produce edges between very similar examples.

Based on these criteria, we chose the following two graph construction methods for our experiments.

MST: Here we simply construct a minimum spanning tree on the entire dataset. This graph is connected, sparse, and furthermore has the appealing property that it has no free parameters to adjust. In addition, because the exact MRF per-node probabilities *can* be exactly calculated on a tree, it allows us to compare our randomized mincut method with the exact MRF calculation.

δ-**MST:** For this method, we connect two points with an edge if they are within a radius δ of each other. We then view the components produced as supernodes and connect them via an MST. Blum and Chawla [12] used δ such that the largest component had half the vertices (but did not do the second MST stage). To produce a more sparse graph, we choose δ so

[4]This is perhaps an obvious criterion but it is important to keep in mind. For instance, if examples are uniform random points in the 1-dimensional interval $[0, 1]$, and we connect each point to its nearest k neighbors, then it is not hard to see that if k is fixed and the number of points goes to infinity, the number of components will go to infinity as well. That is because a local configuration, such as two adjacent tight clumps of k points each, can cause such a graph to disconnect.

that the largest component has $1/4$ of the vertices.

Another natural method to consider would be a k-NN graph, say connected up via a minimum spanning tree as in δ-MST. However, experimentally, we find that on many of our datasets this produces graphs where the mincut algorithm is simply not able to find even moderately balanced cuts (so it ends up rejecting them all in its internal "sanity-check" procedure). Thus, even with a small labeled dataset, the mincut-based procedure would tell us to choose an alternative graph-creation method.

3.6 Experimental Analysis

We compare the randomized mincut algorithm on a number of datasets with the following approaches:

PLAIN MINCUT: Mincut without randomization.

GAUSSIAN FIELDS: The algorithm of Zhu et al.[85].

SGT: The spectral algorithm of Joachims [44].

EXACT: The exact Bayes-optimal prediction in the MRF model, which can be computed efficiently in trees (so we only run it on the MST graphs).

Below we present results on handwritten digits, portions of the 20 newsgroups text collection, and various UCI datasets.

3.6.1 Handwritten Digits

We evaluated randomized mincut on a dataset of handwritten digits originally from the Cedar Buffalo binary digits database [41]. Each digit is represented by a 16 X 16 grid with pixel values ranging from 0 to 255. Hence, each image is represented by a 256-dimensional vector.

For each size of the labeled set, we perform 10 trials, randomly sampling the labeled points

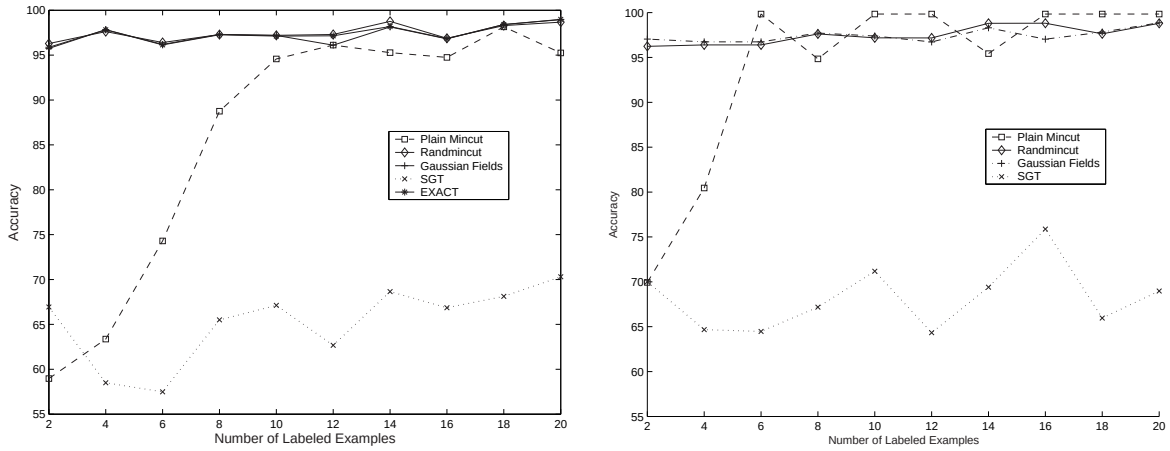

Figure 3.2: "1" vs "2" on the digits dataset with the MST graph (left) and $\delta_{\frac{1}{4}}$ graph (right).

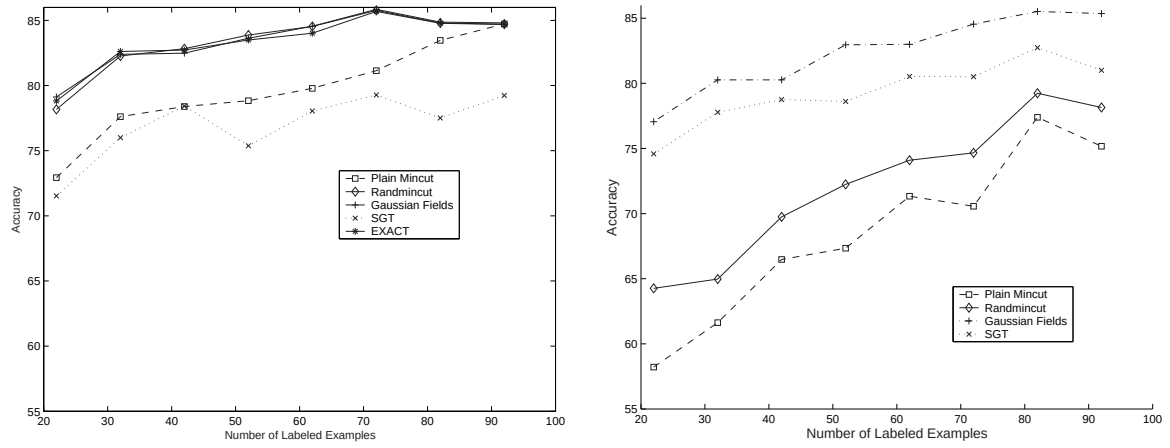

Figure 3.3: Odd vs. Even on the digits dataset with the MST graph (left) and $\delta_{\frac{1}{4}}$ graph (right).

from the entire dataset. If any class is not represented in the labeled set, we redo the sample.

One vs. Two: We consider the problem of classifying digits, "1" vs. "2" with 1128 images. Results are reported in Figure 3.2. We find that randomization substantially helps the mincut procedure when the number of labeled examples is small, and that randomized mincut and the Gaussian field method perform very similarly. The SGT method does not perform very well on this dataset for these graph-construction procedures. (This is perhaps an unfair comparison, because our graph-construction procedures are based on the needs of the mincut algorithm, which may be different than the design criteria one would use for

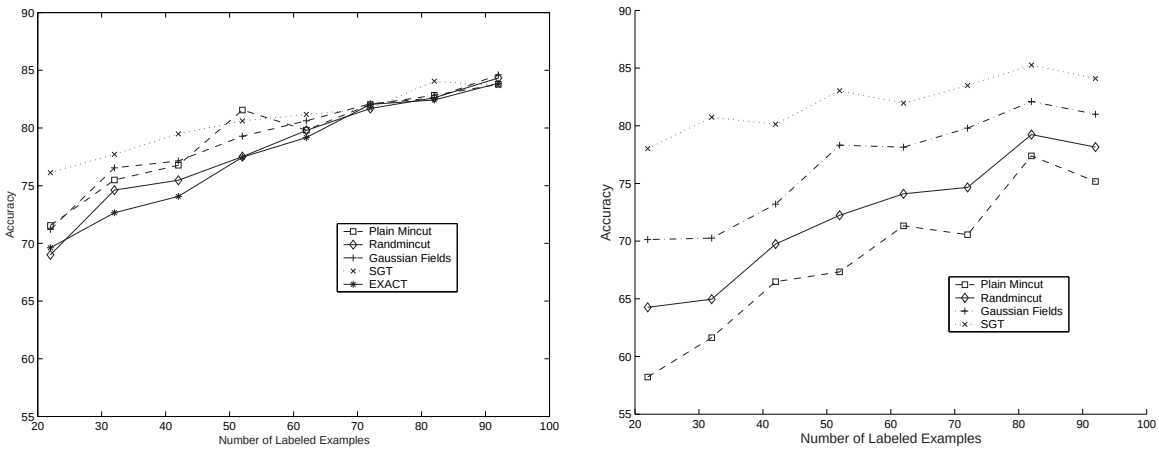

Figure 3.4: PC vs MAC on the 20 newsgroup dataset with MST graph (left) and $\delta_{\frac{1}{4}}$ graph (right).

graphs for SGT.)

Odd vs. Even: Here we classify 4000 digits into Odd vs. Even. Results are given in Figure 3.3. On the MST graph, we find that Randomized mincut, Gaussian fields, and the exact MRF calculation all perform well (and nearly identically). Again, randomization substantially helps the mincut procedure when the number of labeled examples is small. On the δ-MST graph, however, the mincut-based procedures perform substantially worse, and here Gaussian fields and SGT are the top performers.

In both datasets, the randomized mincut algorithm tracks the exact MRF Bayes-optimal predictions extremely closely. Perhaps what is most surprising, however, is how good performance is on the simple MST graph. On the Odd vs. Even problem, for instance, Zhu et al. [85] report for their graphs an accuracy of 73% at 22 labeled examples, 77% at 32 labeled examples, and do not exceed 80% until 62 labeled examples.

3.6.2 20 newsgroups

We performed experiments on classifying text data from the 20-newsgroup datasets, specifically PC versus MAC (see Figure 3.4). Here we find that on the MST graph, all the methods perform similarly, with SGT edging out the others on the smaller labeled set sizes. On the δ-MST graph,

SGT performs best across the range of labeled set sizes. On this dataset, randomization has much less of an effect on the mincut algorithm.

DATASET	$\|L\|\&\|U\|$	FEAT.	GRAPH	MINCUT	RAND MINCUT	GAUSSIAN	SGT	EXACT
			MST	92.0	90.9	90.6	90.0	90.6
VOTING	45+390	16	$\delta_{\frac{1}{4}}$	92.3	91.2	91.0	85.9	—
			MST	86.1	89.4	92.2	89.0	92.4
MUSH	20+1000	22	$\delta_{\frac{1}{4}}$	94.3	94.2	94.2	91.6	—
			MST	78.3	77.8	79.2	78.1	83.9
IONO	50+300	34	$\delta_{\frac{1}{4}}$	78.8	80.0	82.8	79.7	—
			MST	63.5	64.0	63.7	61.8	63.9
BUPA	45+300	6	$\delta_{\frac{1}{4}}$	62.9	62.9	63.5	61.6	—
			MST	65.7	67.9	66.7	67.7	67.7
PIMA	50+718	8	$\delta_{\frac{1}{4}}$	67.9	68.8	67.5	68.2	—

Table 3.1: Classification accuracies of basic mincut, randomized mincut, Gaussian fields, SGT, and the exact MRF calculation on datasets from the UCI repository using the MST and $\delta_{\frac{1}{4}}$ graph.

3.6.3 UCI Datasets

We conducted experiments on various UC Irvine datasets; see Table 3.1. Here we find all the algorithm perform comparably.

3.6.4 Accuracy Coverage Tradeoff

As mentioned earlier, one motivation for adding randomness to the mincut procedure is that we can use it to set a confidence level based on the number of cuts that agree on the classification of a particular example. To see how confidence affects prediction accuracy, we sorted the examples by

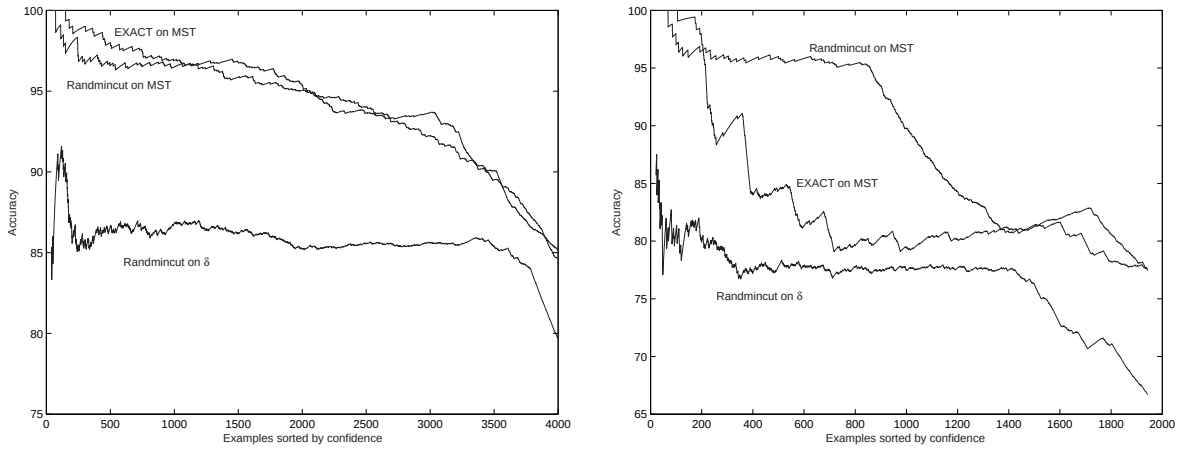

Figure 3.5: Accuracy coverage tradeoffs for randomized mincut and EXACT. Odd vs. Even (left) and PC vs. MAC (right).

confidence and plotted the cumulative accuracy. Figure 3.5 shows accuracy-coverage tradeoffs for Odd-vs-Even and PC-vs-MAC. We see an especially smooth tradeoff for the digits data, and we observe on both datasets that the algorithm obtains a substantially lower error rate on examples on which it has high confidence.

3.6.5 Examining the graphs

To get a feel for why the performance of the algorithms is so good on the MST graph for the digits dataset, we examined the following question. Suppose for some $i \in \{0, \ldots, 9\}$ you remove all vertices that are not digit i. What is the size of the largest component in the graph remaining? This gives a sense of how well one could possibly hope to do on the MST graph if one had only one labeled example of each digit. The result is shown in Figure 3.6. Interestingly, we see that most digits have a substantial fraction of their examples in a single component. This partly explains the good performance of the various algorithms on the MST graph.

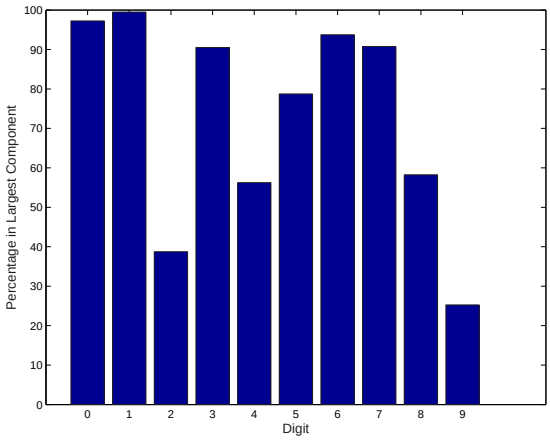

Figure 3.6: MST graph for Odd vs. Even: percentage of digit i that is in the largest component if all other digits were deleted from the graph.

3.7 Conclusion

The randomized mincut algorithm addresses several shortcomings of the basic mincut approach, improving performance especially when the number of labeled examples is small, as well as providing a confidence score for accuracy-coverage curves. We can theoretical motivate this approach from a sample complexity and Markov Random Field perspective.

The experimental results support the applicability of the randomized mincut algorithm to various settings. In the experiments done so far, our method allows mincut to approach, though it tends not to beat, the Gaussian field method of Zhu et al. [85]. However, mincuts have the nice property that we can apply sample-complexity analysis, and furthermore the algorithm can often easily tell when it is or is not appropriate for a dataset based on how large and how unbalanced the cuts happen to be.

The exact MRF per-node likelihoods can be computed efficiently on trees. It would be interesting if this can be extended to larger classes of graphs.

Chapter 4

Local Linear Semi-supervised Regression

4.1 Introduction and Motivation

In many machine learning domains, labeled data is much more expensive than labeled data. For example, labeled data may require a human expert or an expensive experimental process to classify each example. For this reason there has been a lot of interest in the last few years in machine learning algorithms than can make use of unlabeled data [25]. The majority of such proposed algorithms have been applied to the classification task. In this chapter we focus on using unlabeled data in regression. In particular, we present LLSR, the first extension of Local Linear Regression to the problem of semi-supervised learning.

4.1.1 Regression

Regression is a fundamental tool in statistical analysis. At its core regression aims to model the relationship between 2 or more random variables. For example, an economist might want to investigate whether more education leads to an increased income. A natural way to accomplish this is to take number of years of education as the dependent variable and annual income as the

independent variable and to use regression analysis to determine their relationship.

Formally, we are given as input $(X_1, Y_1), ...(X_n, Y_n)$ where the X_i are the independent variables and Y_i are the dependent variables. We want to predict for any X, the value of the corresponding Y. There are two main types of techniques used to accomplish this:

1. Parametric regression: In this case we assume that the relationship between the variable is of a certain type (e.g. a linear relationship) and we are concerned with learning the parameters for a relationship of that type which best fit the data.
2. Non-parametric regression: In this case we do not make any assumptions about the type of relationship that holds between the variables, but we derive this relationship directly from the data.

Regression analysis is heavily used in the natural sciences and in social sciences such as economics, sociology and political science. A wide variety of regression algorithms are used including linear regression, polynomial regression and logistic regression (among the parametric methods) and kernel regression and local linear regression (among the non-parametric methods). A further discussion of such methods can be found in any introductory statistics textbook [77, 78].

In semi-supervised regression in addition to getting the dependent and independent variables X and Y we are also given an addition variable R which indicates whether or not we observe that value of Y. In other words we get data $(X_1, Y_1, R_1), ...(X_n, Y_n, R_n)$ and we observe Y_i only if $R_i = 1$.

We note that the problem of semi-supervised regression is more general than the semi-

supervised classification problem. In the latter case the Y_i are constrained to have only a finite number of possible values whereas in regression the Y_i are assumed to be continuous. Hence some algorithms designed for semi-supervised classification (e.g. graph mincut[12]) are not applicable to the more general semi-supervised regression problem. Other algorithms such as Gaussian Fields [85]) are applicable to both problems.

Although semi-supervised regression has received less attention than semi-supervised classification, a number of methods have been developed dealing specifically with this problem. These include the transductive regression algorithm proposed by Cortes and Mohri [23]and co-training style algorithms proposed by Zhou and Li [82], Sindhwani et al.[68] and Brefeld et al.[17].

4.1.2 Local Linear Semi-supervised Regression

In formulating semi-supervised classification algorithms, an often useful motivating idea is the Cluster Assumption: the assumption that the data will naturally cluster into clumps that have the same label. This notion of clustering does not readily apply to regression, but we can make a somewhat similar "smoothness" assumption: we expect the value of the regression function to not "jump" or change suddenly. In both cases we expect *examples that are close to each other to have similar values.*

A very natural way to instantiate this assumption in semi-supervised regression is by finding estimates $\hat{m}(x)$ that minimize the following objective function (subject to the constraint that $\hat{m}(x_i) = y_i$ for the labeled data):

$$\sum_{i,j} w_{ij}(\hat{m}(x_i) - \hat{m}(x_j))^2$$

where $\hat{m}(x_i)$ is the *estimated* value of the function at example x_i, w_{ij} is a measure of the similarity between examples x_i and x_j and y_i is the value of the function at x_i (only defined on the labeled examples).

This is exactly the objective function minimized by the Gaussian Fields algorithm proposed by Zhu, Gharamani and Lafferty [85].

This algorithm has several attractive properties:

1. The solution can be computed in closed form by simple matrix operations.
2. It has interesting connections to Markov Random Fields, electrical networks,spectral graph theory and random walks. For example, for the case of boolean weights w_{ij}, the estimates $\hat{m}(x_i)$ produced from this optimization can be viewed as the probability that a random walk starting at x_i on the graph induced by the weights, would reach a labeled positive example before reaching a labeled negative example. These connections are further explored in works by Zhu et al. [84, 85].

However, it suffers from at least one major drawback when used in regression: It is "locally constant." This means that it will tend to assign the same value to all the example near a particular labeled example and hence produce "flat neighborhoods."

While this behavior is desirable in classification, it is often undesirable in regression application where we frequently assume that the true function is "locally linear." By locally linear we mean that(on some sufficiently small scale) the value of an example is a "linear interpolation" of the value of its closest neighbors. (From a mathematical point of view local linearity is a consequence of a function being differentiable.)Hence if our function is of "locally linear" type then a "locally constant" estimator will not provide good estimates and we would prefer to use

an algorithm that incorporates a local linearity assumption.

The supervised analogue of Gaussian Fields is Weighted Kernel Regression (also known as the Nadaraya-Watson estimator) which minimizes the following objective function:

$$\sum_i w_i(y_i - \hat{m}(x))^2$$

where $\hat{m}(x)$ is the value of the function at example x,y_i is the value of x_i and w_i is a measure of the similarity between x and x_i.

In the supervised case, there already exists an estimator that has the desired property: Local Linear Regression, which finds β_x so as to minimize the following objective function:

$$\sum_{i=1}^{n} w_i(y_i - \beta_x^T X_{xi})^2$$

with

$$X_{xi} = \begin{pmatrix} 1 \\ x_i - x \end{pmatrix}.$$

Hence, a suitable goal is to derive a local linear version of the Gaussian Fields algorithm. Equivalently, we want a semi-supervised version of the Local Linear estimator.

In the remainder of this chapter we will give some background on non-parametric regression, describe the Local Linear Semi-supervised Regression algorithm and show the results of some experiments on real and synthetic data.

4.2 Background

The general problem of estimating a function from data has been extensively studied in the statistics community. There are two broad classes of methods that are used: Parametric and Non-parametric. We describe these in turn below.

4.2.1 Parametric Regression methods

These approaches assume that the function that is being estimated is of a particular type and then try to estimate the parameters of the function so that it will best fit the observed data.

For example, we may assume that the function we seek is linear but the observations have been corrupted with Gaussian noise:

$$y = \beta^T x + \epsilon_i \text{ (with } \epsilon_i \sim N(0, \sigma^2)).$$

Parametric methods have some advantages compared to non-parametric methods:

1. They are usually easier to analyze mathematically.
2. They usually require less data in order to learn a good model.
3. They are typically computationally less intensive.

However, they also have several disadvantages, especially if the assumptions are not entirely correct.

4.2.2 Non-Parametric Regression methods

These approaches do not assume that the function we are trying to estimate is of a specific type. i.e given

$$y_i = m(x_i) + \epsilon_i$$

the goal is to estimate the value of $m(x)$ at each point.

The main advantage of non-parametric approaches is that they are more flexible than parametric methods and hence they are able to accurately represent broader classes of functions.

4.2.3 Linear Smoothers

Linear smoothers are a class of non-parametric methods in which the function estimates are a linear function of the response variable:

$$\hat{y} = Ly$$

where $\hat{y}$ are the new estimates, y are the observations and L is a matrix which may be constructed based on the data.

Linear smoothers include most commonly used non-parametric regression algorithms and in particular all the algorithms we have discussed so far are linear smoothers. We discuss how we can view each of them as linear smoothers below.

Weighted Kernel Regression

The objective is to find the number $\hat{m}(x)$ that minimizes the least squares error

$$\sum_i w_i(y_i - \hat{m}(x))^2.$$

The minimizer of this objective function is

$$\hat{m}(x) = \frac{\sum_i w_i y_i}{\sum_i w_i} = Ly.$$

Local Linear Regression

The objective is to find β_x that minimizes the least squares error

$$\sum_{i=1}^{n} w_i(y_i - \beta_x^T X_{xi})^2$$

where

$$X_{xi} = \begin{pmatrix} 1 \\ x_i - x \end{pmatrix}.$$

The minimizer of the objective function is

$$\hat{\beta}_x = (X_x^T W_x X_x)^{-1} X_x^T W_x y$$

where X_x is the $n \times (d+1)$ matrix (X_{xi}^T) and the matrix W_x is the $n \times n$ diagonal matrix $diag(w_i)$.

The local linear estimate of $m(x)$ is

$$\hat{m}(x) = e_1^T (X_x^T W_x X_x)^{-1} X_x^T W_x y = Ly.$$

Gaussian Fields

The objective is to find f that minimizes the energy functional

$$\mathcal{E}(f) = \sum_{i,j} w_{ij}(f_i - f_j)^2$$

with the constraint that some of the f_i are fixed.

It can be shown that

$$\mathcal{E}(f) = f^T \Delta f$$

where $\Delta = D - W$ is the *combinatorial graph Laplacian* of the data, W is the weight matrix and D is the diagonal matrix with $D_{ii} = \sum_j w_{ij}$.

If f_L denotes the observed labels and f_U denotes the unknown labels then the minimizer of the energy functional is

$$f_U = \Delta_{UU}^{-1} \Delta_{UL} f_L = Ly$$

where Δ_{UU} and Δ_{UL} are the relevant submatrices of the graph Laplacian.

4.3 Local Linear Semi-supervised Regression

Our goal is to derive a semi-supervised analogue of Local Linear Regression so we want it to have the following properties.

1. It should fit a linear function at each point like Local Linear Regression.
2. The estimate for a particular point should depend on the estimates for all the other examples like in Gaussian Fields. In particular we want to enforce some smoothness in how the linear function changes.

More specifically, let X_i and X_j be two examples and β_i and β_j be the local linear fits at X_i and X_j respectively.

Let

$$X_{ji} = \begin{pmatrix} 1 \\ X_i - X_j \end{pmatrix}.$$

Then $X_{ji}^T \beta_j$ is the estimated value at X_i using the local linear fit at X_j. Thus the quantity

$$(\beta_{i0} - X_{ji}^T \beta_j)^2$$

is the squared difference between the smoothed estimate at X_i and the estimated value at X_i using the local fit at X_j. This situation is illustrated in figure 4.1

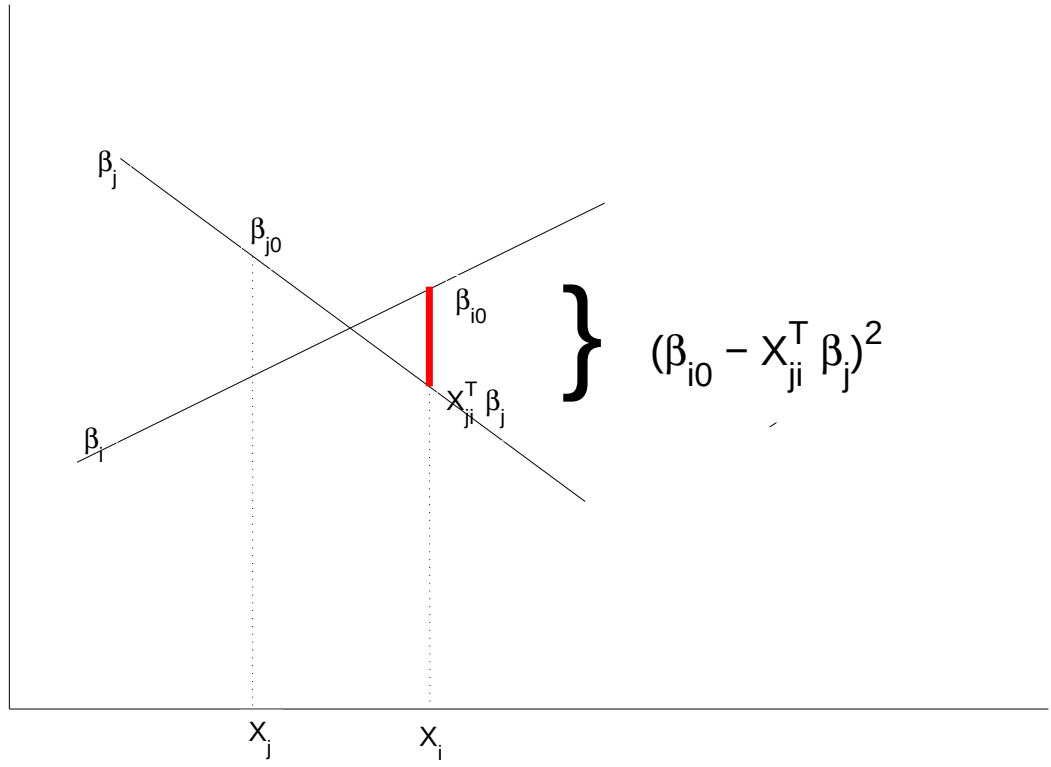

Figure 4.1: We want to minimize the squared difference between the smoothed estimate at X_i and the estimated value at X_i using the local fit at X_j

We can take the sum of this quantity over all pairs of examples as the quantity we want to minimize:

$$\Omega(\beta) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(\beta_{i0} - X_{ji}^T\beta_j)^2$$

with the constraint that some of the B_i are fixed.

Lemma 1.1 The manifold regularization functional $\Omega(\beta)$ can be written as the quadratic form

$$\Omega(\beta) = \beta^T \Delta \beta$$

where the local linear Laplacian $\Delta = [\Delta_{ij}]$ is the $n \times n$ block matrix with $(d+1) \times (d+1)$ blocks $\Delta_{ij} = diag(D_i) - [W_{ij}]$ where

$$D_i = \frac{1}{2}\sum_{j} w_{ij}(e_1 e_1^T + X_{ij}X_{ij}^T)$$

and

$$W_{ij} = w_{ij}\begin{pmatrix} 1 & (X_j^T - X_i^T) \\ (X_j - X_i) & 0 \end{pmatrix}.$$

Proof:

Let n be the number of examples.

Let d be the number of dimensions.

Let X be a $d \times n$ matrix (the data).

Let W be a ***symmetric*** $n \times n$ matrix. (the similarity between X_i and X_j)

Let β be a $n \times (d+1)$ length vector. (the coefficients we want to learn).

Let β_i be the β_{id+1} to $\beta_{i(d+1)}$ coefficients in β. (the coefficients for X_i)

Let X_i be $[X_{i1} \ \dots \ X_{id}]^T$ (The i^{th} column of X)

Let X_{ij} be $[1 \ (X_i - X_j)^T]^T$

Let e_1 be $[1 \ 0 \ 0 \dots \ 0]^T$

Let d_i be $\sum_j W_{ij}$

Starting with the objective function:

$$\Omega(\beta) = \sum_i \sum_j W_{ij}(X_{ii}^T B_i - X_{ij}^T B_j)^2 \text{ (by definition)}$$

We first expand the expression to get:

$$= \sum_i \sum_j W_{ij} B_i^T X_{ii} X_{ii}^T B_i - \sum_i \sum_j 2W_{ij} B_i^T X_{ii} X_{ij}^T B_j + \sum_i \sum_j W_{ij} B_j^T X_{ij} X_{ij}^T B_j \text{ (expanding)}$$

Taking the first term we note that:

$$\sum_i \sum_j W_{ij} B_i^T X_{ii} X_{ii}^T B_i = \sum_i B_i^T d_i X_{ii} X_{ii}^T B_i = \mathcal{B}^T \Delta_1 \mathcal{B}$$

Where $[(\Delta_1)_{ii}] = d_i X_{ii} X_{ii}^T$

Next looking at the second term we get:

$$S = \sum_i \sum_j 2W_{ij} B_i^T X_{ii} X_{ij}^T B_j = \sum_i \sum_j 2B_i^T W_{ij} X_{ii} X_{ij}^T B_j$$

$$= \sum_i \sum_j 2B_j^T W_{ji} X_{jj} X_{ji}^T B_i = \sum_i \sum_j 2B_i^T W_{ij} X_{ji} X_{jj}^T B_j$$

So we can derive the following expression for the second term:

$$S = \frac{1}{2}(S + S) = \frac{1}{2} \sum_i \sum_j 2B_i^T W_{ij} (X_{ii} X_{ij}^T + X_{ji} X_{jj}^T) B_j = \mathcal{B}^T \Delta_2 \mathcal{B}$$

Where $[(\Delta_2)_{ij}] = W_{ij}(X_{ii}X_{ij}^T + X_{ji}X_{jj}^T)$

Finally looking at the third term in the original expression:

$$\sum_i \sum_j W_{ij} B_j^T X_{ij} X_{ij}^T B_j = \sum_j B_j^T (\sum_i W_{ij} X_{ij} X_{ij}^T) B_j = \mathcal{B}^T \Delta_3 \mathcal{B}$$

Where $[(\Delta_3)_{ii}] = \sum_j W_{ij} X_{ji} X_{ji}^T$

So in conclusion:

$$\Omega(\beta) = \sum_i \sum_j W_{ij} (X_{ii}^T B_i - X_{ij}^T B_j)^2 = \mathcal{B}^T \Delta \mathcal{B}$$

where $\Delta = diag(D_i) - [W_{ij}]$ and

$$diag(D_i) = \Delta_1 + \Delta_3$$

$$[W_{ij}] = \Delta_2$$

■

The term we just derived is the local linear manifold regularizer. Now we add another term to account for the labeled examples and minimize the sum.

Lemma 1.2.

Let $\mathcal{R}_\gamma(\beta)$ be the manifold regularized risk functional:

$$\mathcal{R}_\gamma(\beta) = \frac{1}{2}\sum_{j=1}^{n}\sum_{R_i=1} w_{ij}(Y_i - X_{ji}^T\beta_j)^2 + \frac{\gamma}{2}\Omega(\beta)$$

Here $R_i = 1$ means i is a labeled example and γ is a regularization constant. We can simplify this to:

$$= \frac{1}{2}\sum_{j=1}^{n}(Y_i - X_j\beta_j)^T W_j(Y_i - X_j\beta_j) + \frac{\gamma}{2}\beta^T\Delta\beta$$

The minimizer of this risk can be written in closed form as:

$$\hat{\beta}(\gamma) = (diag(X_j^T W_j X_j) + \gamma\Delta)^{-1})(X_1^T W_1 Y, \ldots, X_1^T W_1 Y)^T$$

Proof.

The expression we want to minimize is:

$$\frac{1}{2}\sum_{j=1}^{n}\sum_{R_i=1} W_{ij}(Y_i - X_{ji}^T B_j)^2 + \frac{\gamma}{2}\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}(B_{i0} - X_{ji}^T B_j)^2$$

Using the previous lemma this is equivalent to:

$$= \frac{1}{2}\sum_{j=1}^{n}(Y - X_j B_j)^T W_j(Y - X_j B_j) + \frac{\gamma}{2}\mathcal{B}^T\Delta\mathcal{B}$$

We can expand the first term:

$$= \frac{1}{2}\sum_{j=1}^{n} Y^T W_j Y - \frac{1}{2}\sum_{j=1}^{n} B_j^T X_j^T W_j Y - \frac{1}{2}\sum_{j=1}^{n} Y^T W_j X_j B_j + \frac{1}{2}\sum_{j=1}^{n} B_j^T X_j^T W_j X_j B_j + \frac{\gamma}{2}\mathcal{B}^T\Delta\mathcal{B}$$

After some rearrangement we get:

$$= \frac{1}{2}Y^T(\sum_{j=1}^{n} W_j)Y - B^T[X_j^T W_j Y] + \frac{1}{2}\mathcal{B}^T diag(X_j^T W_j X_j)\mathcal{B} + \frac{\gamma}{2}\mathcal{B}^T\Delta\mathcal{B}$$

Now we let $Q = diag(X_j^T W_j X_j), P = [X_j^T W_j Y], C = \frac{1}{2} Y^T (\sum_{j=1}^{n} W_j) Y$

The expression now becomes:

$$= C - \mathcal{B}^T P + \frac{1}{2}\mathcal{B}^T Q\mathcal{B} + \frac{\gamma}{2}\mathcal{B}^T \Delta\mathcal{B}$$

Now we differentiate with respect to $\mathcal{B}$ and set to zero:

$$= -P + Q\mathcal{B} + \gamma\Delta\mathcal{B} = 0$$

which leads to:

$$=> \mathcal{B} = (Q + \gamma\Delta)^{-1} P$$

So in conclusion the expression which minimizes the manifold regularized risk functional is:

$$\hat{\beta}(\gamma) = (diag(X_j^T W_j X_j) + \gamma\Delta)^{-1})(X_1^T W_1 Y, \ldots, X_1^T W_1 Y)^T$$

■

4.4 An Iterative Algorithm

As defined here the LLSR algorithm requires inverting a $n(d+1) \times n(d+1)$ matrix. This may be impractical if n and d are large. For example for $n = 1500$, $d = 199$ the closed form computation would require inverting a $300,000 \times 300,000$ matrix, a matrix that would take roughly $720\ GB$ of memory to store in Matlab's standard double precision format.

Hence, it is desirable to have a more memory efficient method for computing LLSR. An iterative algorithm can fulfill this requirement.

Theorem 1.3 If we initially assign B_i arbitrary values for all i, and repeatedly apply the following formula, then the B_i will converge to the minimum of the LLSR objective function:

$$B_i = [\sum_{R_j=1} W_{ij}X_{ji}X_{ji}^T + \gamma \sum_j W_{ij}(X_{ii}X_{ii}^T + X_{ji}X_{ji}^T)]^{-1}(\sum_{R_j=1} W_{ij}X_{ji}Y_j + \gamma \sum_j W_{ij}(X_{ii}X_{ij}^T + X_{ji}X_{jj}^T)B_j)$$

Proof.

The objective function is:

$$\frac{1}{2}\sum_{j=1}^{n}\sum_{R_i=1} W_{ij}(Y_i - X_{ij}^T B_j)^2 + \frac{\gamma}{2}\frac{1}{2}\sum_i \sum_j W_{ij}(X_{ii}^T B_i - X_{ij}^T B_j)^2$$

If we differentiate w.r.t to B_i and set to 0 we get:

$$[\sum_{R_j=1} W_{ij}X_{ji}X_{ji}^T + \gamma \sum_j W_{ij}(X_{ii}X_{ii}^T + X_{ji}X_{ji}^T)]B_i = \sum_{R_j=1} W_{ij}X_{ji}Y_j + \gamma \sum_j W_{ij}(X_{ii}X_{ij}^T + X_{ji}X_{jj}^T)B_j$$

After rearranging:

$$\Rightarrow B_i = [\sum_{R_j=1} W_{ij}X_{ji}X_{ji}^T + \gamma \sum_j W_{ij}(X_{ii}X_{ii}^T + X_{ji}X_{ji}^T)]^{-1}(\sum_{R_j=1} W_{ij}X_{ji}Y_j + \gamma \sum_j W_{ij}(X_{ii}X_{ij}^T + X_{ji}X_{jj}^T)B_j)$$

Hence the iterative algorithm is equivalent to doing "exact line search" for each B_i. In other words, given that all other variables are constant, we find the optimal value of B_i so as to minimize the objective function.

This means that at each step the value of the objective function must decrease. But since the objective function is a sum of squares, it can never be less than 0. Hence the iteration will

eventually converge to a local minima. If we further note that the objective function is convex, then it only has one global minimum and that will be the only fixed point of the iteration.Hence the iteration will converge to the global minimum of the objective function. ■

As we noted previously, if $n = 1500$, $d = 199$ the closed form computation would require $720\ GB$ of memory but the iterative computation only requires keeping a vector of length $n \times (d+1)$ and inverting a $(d+1) \times (d+1)$ matrix which in this case only takes 2.4 MB of memory. So we save a factor of almost 300,000 in memory usage in this example.

4.5 Experimental Results

To understand the behavior of the algorithm we performed some experiments on both synthetic and real data.

4.5.1 Algorithms

We compared two purely supervised algorithms with Local Linear Semi-supervised Regression.

WKR - Weighted Kernel Regression

LLR - Local Linear Regression

LLSR - Local Linear Semisupervised Regression

4.5.2 Parameters

There are two free parameters that we have to set for LLSR (and one (kernel bandwidth) for WKR and LLR).

h - kernel bandwidth

γ - Amount of Semisupervised Smoothing

4.5.3 Error metrics

LOOCV MSE - Leave-One-Out-Cross-Validation Mean Squared Error. This is what we actually try to optimize (analogous to training error).

MSE - This is the true Mean Squared Error between our predictions and the true function (analogous to test error).

4.5.4 Computing LOOCV

Since we do not have access to the MSE we pick the parameters so as to minimize the LOOCV MSE. Computing the LOOCV MSE in a naïve way would be very computationally expensive since we would have to run the algorithm $O(n)$ times.

Fortunately for a linear smoother we can compute the LOOCV MSE by running the algorithm only once. More precisely if $\hat{y} = Ly$ then

$$\text{LOOCV MSE} = \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - L_{ii}} \right)^2$$

4.5.5 Automatically selecting parameters

We experimented with a number of different ways of picking the parameters. In these experiments we used a form of coordinate descent.

Picking one parameter

To pick one parameter we just reduce the bandwidth until there is no more improvement in LOOCV MSE.

1. Initially set bandwidth to 1 and compute LOOCV MSE.
2. Set $h = h/2$ and compute the resulting LOOCV MSE.
3. If the LOOCV MSE decreases then go back to step 2 else go to step 4
4. Output the h which had the lowest LOOCV MSE.

Picking two parameters

To pick two parameters we succesively halve the parameter which yields the biggest decrease in LOOCV MSE.

1. Initially set both bandwidth and smoothing parameter to 1 and compute LOOCV MSE.
2. Set $h = h/2$ while leaving γ alone and compute LOOCV MSE.
3. Set $\gamma = \gamma/2$ while leaving h alone and compute LOOCV MSE.
4. If either steps 2 or 3 decreased the LOOCV MSE then choose the setting which had the lower LOOCV MSE and go back to step 2 else go to step 5.
5. Output the parameter setting which had the lowest MSE.

These procedures are a crude form of gradient descent.

Although they are not guaranteed to be optimal they are (somewhat) effective in practice.

4.5.6 Synthetic Data Set: Gong

The Gong function is a popular function for testing regression algorithms.

Interval: 0.5 to 1.5

Data Points: Sampled uniformly in the interval. (Default $= 800$)

Labeled Data Points: Sampled uniformly from the data points. (Default $= 80$).

RED = Estimated values

BLACK = Labeled examples

True function: $y = \frac{1}{x} \sin \frac{15}{x}$

$\sigma^2 = 0.1$ (Noise)

Figures 4.2, 4.3 and 4.4 show the results of running WKR, LLR and LLSR respectfully on this example. A table summarizing the results is given below in table 4.1. As can be seen LLSR performs substantially better than the other methods in its MSE and from the figures one can see that solution produced by LLSR is much smoother than the others.

Algorithm	MSE
WKR	25.67
LLR	14.39
LLSR	7.99

Table 4.1: Performance of different algorithms on the Gong dataset

Weighted kernel Regression

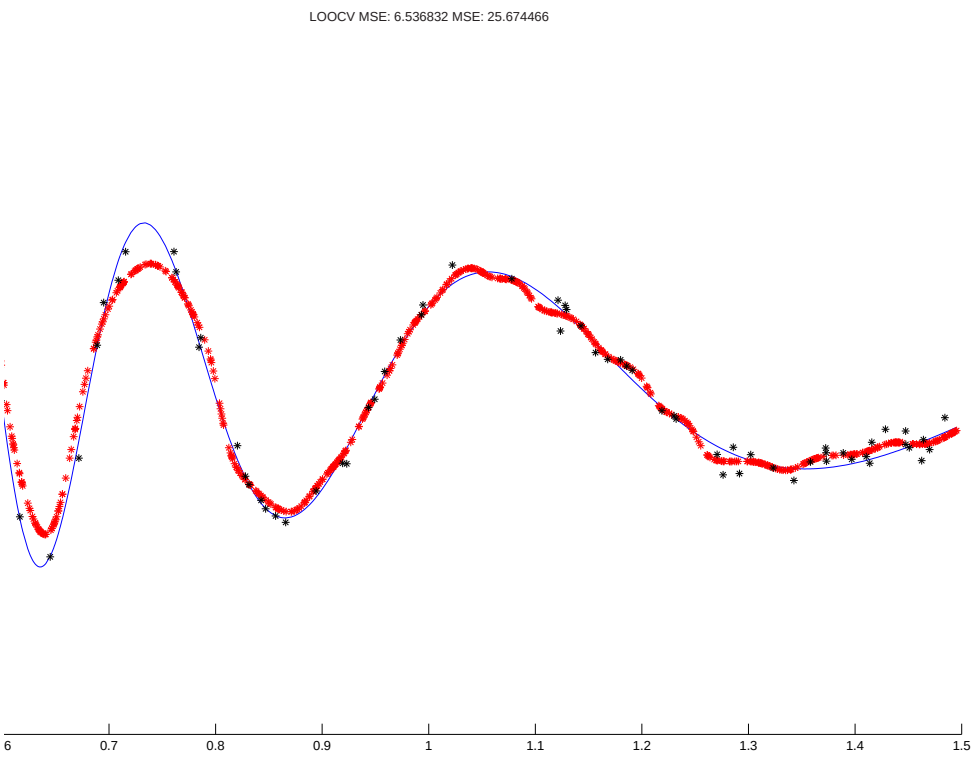

Figure 4.2: WKR on the Gong example, $h = \frac{1}{128}$

Discussion

There is significant bias on the left boundary and at the peaks and valleys. In this case it seems like a local linear assumption might be more appropriate.

Local Linear Regression

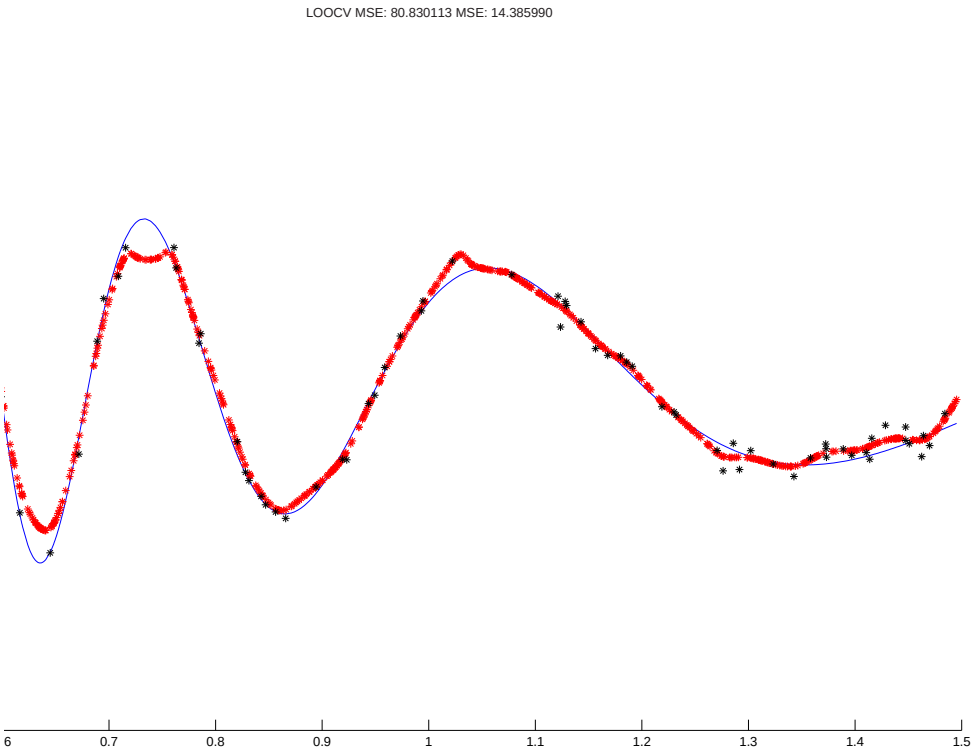

Figure 4.3: LLR on the Gong example, $h = \frac{1}{128}$

Discussion

There is less bias on the left boundary but there seems to be over fitting at the peaks. It seems like more smoothing is needed.

Local Linear Semi-supervised Regression

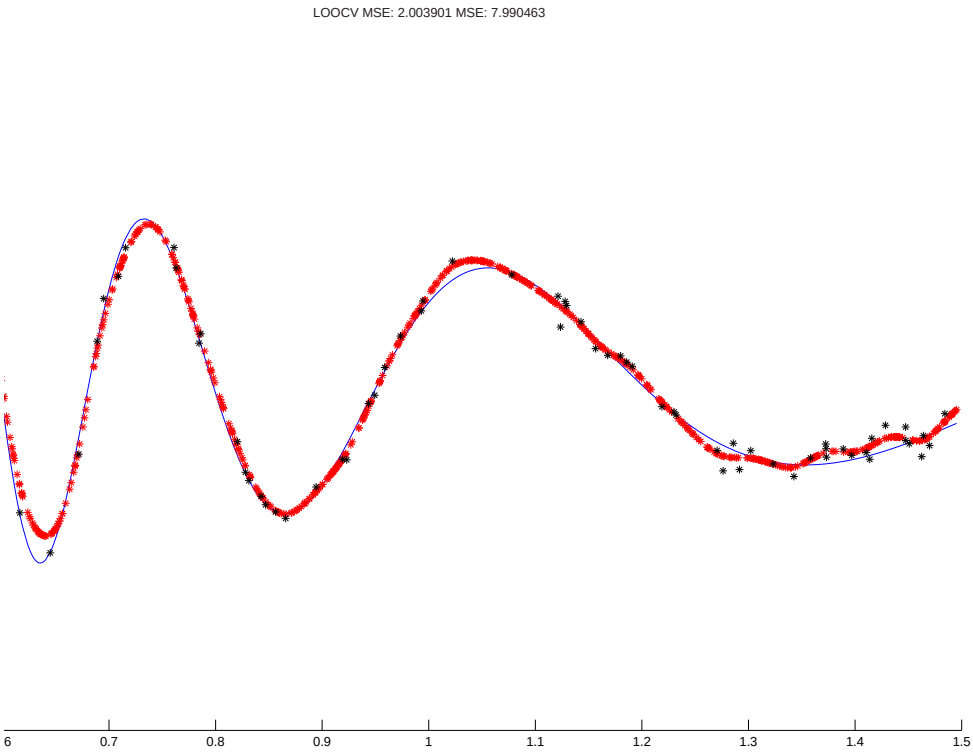

Figure 4.4: LLSR on the Gong example, $h = \frac{1}{512}, \gamma = 1$

Discussion

Although the fit is not perfect, it is the best of the 3. It manages to avoid boundary bias and fits most of the peaks and valleys.

4.5.7 Local Learning Regularization

Recently Schölkopf and Wu [63] have proposed Local Learning Regularization (LL-Reg) as a semi-supervised regression algorithm. They also propose a flexible framework that generalizes many of the well known semi-supervised learning algorithms.

Suppose we can cast our semi-supervised regression problem as finding the f that minimizes the following objective function:

$$f^T R f + (f - y)^T C (f - y)$$

We can easily see that the f that minimizes this objective function is

$$f = (R + C)^{-1} C y$$

The first term is the "semi-supervised" component and imposes some degree of "smoothness" on the predictions. The second term is the "supervised" part indicating the agreement of the predictions with the labeled examples. By choosing different matrices R and C we obtain different algorithms. Typically C is chosen to be the identity matrix so we focus on the choice of R.

It turns out that popular semi-supervised learning algorithms such as the harmonic algorithm [85] and NLap-Reg [81] can be cast in this framework with an appropriate choice of R. For example to get the harmonic algorithm of Zhu, Gharamani and Lafferty [85]we choose R to be the combinatorial graph Laplacian. To get the NLap-Reg algorithm of Zhou et al. [81] we choose R to be the normalized graph Laplacian.

Schölkopf and Wu [63] propose to use the following as the first term in the objective function

$$\sum (f_i - o_i(x_i))^2$$

where $o_i(x_i)$ is the local prediction for the value of x_i based on the value of its neighbors. Again we can choose different functions for the local predictor $o(x)$ and get correspondingly distinct algorithms.

Key point: If the local prediction at x_i is a **linear combination** of the value of its neighbors then we can write $\sum (f_i - o_i(x_i))^2$ as $f^T R f$ for some suitable R.

To see this note that

$$\sum (f_i - o_i(x_i))^2 = ||f - o||^2$$

But if each prediction is a linear combination then $o = Af$ (for some matrix A) and

$$||f - o||^2 = ||f - Af||^2 = ||(I - A)f||^2 = f^T (I - A)^T (I - A) f$$

Hence $R = (I - A)^T (I - A)$.

So the only thing we have to do is pick the function $o_i(x_i)$ then the R will be fixed.

Schölkopf and Wu [63] propose using kernel ridge regression as the local predictor. This will tend to enforce a roughly linear relationship between the predictors. This makes LL-Reg a good candidate to compare against LLSR.

4.5.8 Further Experiments

To gain a better understanding of the performance of LLSR we also compare with (LL-Reg) in addition to WKR and LLR on some real world datasets. The number of examples(*n*), dimen-

sions(*d*) and number of labeled examples(*nl*) in each dataset are indicated in tables 4.2 and 4.3.

Procedure

For each dataset we select a random labeled subset, select parameters using cross validation and compute the root mean squared error of the predictions on the unlabeled data. We repeat this 10 times and report the mean and standard deviation. We also report OPT, the root mean squared error of selecting the optimal parameters in the search space.

Model Selection

For model selection we do a grid search in the parameter space for the best Leave-One-Out Cross Validation rms error on the unlabeled data.

We also report the rms error for the optimal parameters within the range.

For LLSR we search over $\gamma \in \{\frac{1}{100}, \frac{1}{10}, 1, 10, 100\}, h \in \{\frac{1}{100}, \frac{1}{10}, 1, 10, 100\}$

For LL-Reg we search over $\lambda \in \{\frac{1}{100}, \frac{1}{10}, 1, 10, 100\}, h \in \{\frac{1}{100}, \frac{1}{10}, 1, 10, 100\}$

For WKR we search over $h \in \{\frac{1}{100}, \frac{1}{10}, 1, 10, 100\}$

For LLR we search over $h \in \{\frac{1}{100}, \frac{1}{10}, 1, 10, 100\}$

Results

Dataset	n	d	nl	LLSR	LLSR-OPT	WKR	WKR-OPT
Carbon	58	1	10	27±25	19±11	70±36	37±11
Alligators	25	1	10	288±176	209±162	336±210	324±211
Smoke	25	1	10	82±13	79±13	83±19	80±15
Autompg	392	7	100	50±2	49±1	57±3	57±3

Table 4.2: Performance of LLSR and WKR on some benchmark datasets

Dataset	n	d	nl	LLR	LLR-OPT	LL-Reg	LL-Reg-OPT
Carbon	58	1	10	57±16	54±10	162±199	74±22
Alligators	25	1	10	207±140	207±140	289±222	248±157
Smoke	25	1	10	82±12	80±13	82±14	70±6
Autompg	392	7	100	53±3	52±3	53±4	51±2

Table 4.3: Performance of LLR and LL-Reg on some benchmark datasets

4.6 Discussion

From these results combined with the synthetic experiments, LLSR seems to be most helpful on one dimensional datasets which have a “smooth” curve. The Carbon dataset happens to be of this type and LLSR performs particularly well on this dataset. On the other datasets performs competitively but not decisively better than the other algorithms. This is not surprising give the motivation behind the design of LLSR which was to smooth out the predictions, hence LLSR is likely to be more successful on datasets which meet this assumption.

4.7 Conclusion

We introduce Local Linear Semi-supervised Regression and show that it can be effective in taking advantage of unlabeled data. In particular, LLSR seems to perform somewhat better than WKR and LLR at fitting “peaks” and “valleys” where there are gaps in the labeled data. In general if the gaps between labeled data are not too big and the true function is “smooth” LLSR seems to achieve a lower true Mean Squared Error than the purely supervised algorithms.

Chapter 5

Learning by Combining Native Features with Similarity Functions

In this report we describe a new approach to learning with labeled and unlabeled data using similarity functions together with native features, inspired by recent theoretical work [2, 4]. In the rest of this report we will describe some motivations for learning with similarity functions, give some background information, describe our algorithms and present some experimental results on both synthetic and real examples. We give a method that given any pairwise similarity function (which need not be symmetric or positive definite as with kernels) can use unlabeled data to augment a given set of features in a way that allows a learning algorithm to exploit the best aspects of both. We also give a new, useful method for constructing a similarity function from unlabeled data.

5.1 Motivation

Two main motivations for learning with similarity functions are (1) Generalizing and Understanding Kernels and (2) Combining Graph Based or Nearest Neighbor Style algorithms with Feature Based Learning algorithms. We will expand on both of these below.

5.1.1 Generalizing and Understanding Kernels

Since the introduction of Support Vector Machines [62, 64, 65] in the mid 90s, kernel methods have become extremely popular in the machine learning community. This popularity is largely due to the so-called "kernel trick" which allows kernelized algorithms to operate in high dimensional spaces without incurring a corresponding computational cost. The ideas is that if data is not linearly separable in the original feature space kernel methods may be able to find a linear separator in some high dimensional space without too much extra computational cost. And furthermore if data is separable by a large margin then we can hope generalize well from not too many labeled examples.

However, in spite of the rich theory and practical applications of kernel methods, there are a few unsatisfactory aspects. In machine learning applications the intuition behind a kernel is that they serve as a measure of similarity between two objects. However, the theory of kernel methods talks about finding linear separators in high dimensional spaces that we may not even be able to calculate much less understand. This disconnect between the theory and practical applications makes it difficult to gain theoretical guidance in choosing good kernels for particular problems.

Secondly and perhaps more importantly, kernels are required to be symmetric and positive-semidefinite. The second condition in particular is not satisfied by many practically useful similarity functions(for example the Smith-Waterman score in computational biology [76]). In fact, in Section 5.3.1 we give a very natural and useful similarity function that does not satisfy either condition. Hence if these similarity functions are to be used with kernel methods, they have to be coerced into a "legal" kernel. Such coercion may substantially reduce the quality of the similarity functions.

From such motivations, Balcan and Blum [2, 4] recently initiated the study of general simi-

larity functions. Their theory gives a definition of a similarity function that has standard kernels as a special case and they show how it is possible to learn a linear separator with a similarity function and give similar guarantees to those that are obtained with kernel methods.

One interesting aspect of their work is that they give a prominent role to unlabeled data. In particular unlabeled data is used in defining the mapping that projects the data into a linearly separable space. This makes their technique very practical since unlabeled data is usually available in greater quantities than labeled data in most applications.

The work of Balcan and Blum provides a solid theoretical foundation, but its practical implications have not yet been fully explored. Practical algorithms for learning with similarity functions could be useful in a wide variety of areas, two prominent examples being bioinformatics and text learning. Considerable effort has been expended in developing specialized kernels for these domains. But in both cases, it is easy to define similarity functions that are not legal kernels but match well with our desired notions of similarity (for an example in bioinformatics see Vert et al. [76]).

Hence, we propose to pursue a practical study of learning with similarity functions. In particular we are interested in understanding the conditions under which similarity functions can be practically useful and developing techniques to get the best performance when using similarity functions.

5.1.2 Combining Graph Based and Feature Based learning Algorithms.

Feature-based and Graph-based algorithms form two of the dominant paradigms in machine learning. Feature-based algorithms such as Decision Trees[56], Logistic Regression[51], Winnow[53],

and others view their input as feature vectors and use feature values directly to make decisions. Graph-based algorithms, such as the semi-supervised algorithms of [6, 12, 14, 44, 63, 84, 85], instead view examples as nodes in a graph for which the only information available about them is their pairwise relationship (edge weights) to other nodes in the graph. Kernel methods [62, 64, 65, 66] can also be viewed in a sense as graph-based approaches, thinking of $\mathbb{K}(x, x')$ as the weight of edge (x, x').

Both types of approaches have been highly successful, though they each have their own strengths and weaknesses. Feature-based methods perform particularly well on text data, for instance, where individual keywords or phrases can be highly predictive. Graph-based methods perform particularly well in semi-supervised or transductive settings, where one can use similarities to unlabeled or future data, and reasoning based on transitivity (two examples similar to the same cluster of points, or making a group decision based on mutual relationships) in order to aid in prediction. However, they each have weaknesses as well: graph-based (and kernel-based) methods encode all their information about examples into the pairwise relationships between examples, and so they lose other useful information that may be present in features. Feature-based methods have trouble using the kinds of "transitive" reasoning made possible by graph-based approaches.

It turns out again, that similarity functions provide a possible method for combining these two disparate approaches. This idea is also motivated by the same work of Balcan and Blum[2, 4] that we have referred to previously. They show that given a pairwise measure of similarity $\mathbb{K}(x, x')$ between data objects, one can essentially construct features in a straightforward way by collecting a set $x_1, \ldots, x_n$ of random unlabeled examples and then using $\mathbb{K}(x, x_i)$ as the i^{th} feature of example x. They show that if $\mathbb{K}$ was a large-margin kernel function then with high probability the data will be approximately linearly separable in the new space. So our approach to combining

graph based and feature based methods is to keep the original features and augment them (rather than replace them) with the new features obtained by the Balcan-Blum approach.

5.2 Background

We now give background information on algorithms that rely on finding large margin linear separators, kernels and the kernel trick and the Balcan-Blum approach to learning with similarity functions.

5.2.1 Linear Separators and Large Margins

Machine learning algorithms based on linear separators attempt to find a hyperplane that separates the positive from the negative examples; i.e if example x has label $y \in \{+1, -1\}$ we want to find a vector w such that $y(w \cdot x) > 0$.

Linear separators are currently among the most popular machine learning algorithms, both among practitioners and researchers. They have a rich theory and have been shown to be effective in many applications. Examples of linear separator algorithms are Perceptron[56], Winnow[53] and SVM [62, 64, 65]

An important concept in linear separator algorithms is the notion of "margin." Margin is considered a property of the dataset and (roughly speaking) represents the "gap" between the positive and negative examples. Theoretical analysis has shown that the performance of a linear separator algorithm is directly proportional to the size of the margin (the larger the margin the better the performance). The following theorem is just one example of this kind of result:

Theorem In order to achieve error ϵ with probability at least $1 - \delta$, it suffices for a linear separator algorithm to find a separator of margin at leastγ on a dataset of size

$$O(\frac{1}{\epsilon}[\frac{1}{\gamma^2}\log^2(\frac{1}{\gamma\epsilon}) + \log(\frac{1}{\delta})]).$$

Here, the margin is defined as the minimum distance of examples to the separating hyperplane if all examples are normalized to have length at most 1. This bound makes clear the dependence on γ, i.e as the margin gets larger, substantially fewer examples are needed [15, 66] .

5.2.2 The Kernel Trick

A kernel is a function $\mathbb{K}(x, y)$ which satisfies certain conditions:

1. continuous
2. symmetric
3. positive semi-definite

If these conditions are satisfied then Mercer's theorem [55] states that $\mathbb{K}(x, y)$ can be expressed as a dot product in a high-dimensional space i.e there exists a function $\Phi(x)$ such that

$$\mathbb{K}(x, y) = \Phi(x) \cdot \Phi(y)$$

Hence the function $\Phi(x)$ is a mapping from the original space into a new possibly much higher dimensional space. The "kernel trick" is essentially the fact that we can get the results of this high dimensional inner product without having to explicitly construct the mapping $\Phi(x)$. The dimension of the space mapped to by Φ might be huge, but the hope is the margin will be large so we can apply the theorem connecting margins and learnability.

5.2.3 Kernels and the Johnson-Lindenstrauss Lemma

The Johnson-Lindenstrauss Lemma[29] states that a set of n points in a high dimensional Euclidean space can be mapped down into an $O(\log n/\epsilon^2)$ dimensional Euclidean space such that the distance between any two points changes by only a factor of $(1 \pm \epsilon)$.

Arriaga and Vempala [1] use the Johnson-Lindenstrauss Lemma to show that a random linear projection from the ϕ-space to a space of dimension $\tilde{O}(1/\gamma^2)$ approximately preserves linear separability. Balcan, Blum and Vempala [4] then give an explicit algorithm for performing such a mapping. An important point to note is that their algorithm requires access to the distribution where the examples come from in the form of unlabeled data. The upshot is that instead of having the linear separator live in some possibly infinite dimensional space, we can project it into a space whose dimension depends on the margin in the high-dimensional space and where the data is linearly separable if it was linearly separable in the high dimensional space.

5.2.4 A Theory of Learning With Similarity Functions

The mapping discussed in the previous section depended on $\mathbb{K}(x, y)$ being a legal kernel function. In [2] Balcan and Blum show that it is possible to use a similarity function which is not necessarily a legal kernel in a similar way to explicitly map the data into a new space. This mapping also makes use of unlabeled data.

Furthermore, similar guarantees hold: If the data was separable by the similarity function with a certain margin then it will be linearly separable in the new space. The implication is that any valid similarity function can be used to map the data into a new space and then a standard linear separator algorithm can be used for learning.

5.2.5 Winnow

Now we make a slight digression to describe the algorithm that we will be using. Winnow is an online learning algorithm proposed by Nick Littlestone [53]. Winnow starts out with a set of weights and updates them as it sees examples one by one using the following update procedure:

Given a set of weights $w = \{w_1, w_2, w_3, \ldots w_d\} \in \mathbb{R}^d$ and an example $\{x = \{x_1, x_2, x_3, \ldots x_d\} \in \{0,1\}^d\}$

1. If ($w \cdot x \geq d$) then set $y_{pred} = 1$ else set $y_{pred} = 0$. Output y_{pred} as our prediction.

2. Observe the true label $y \in \{0,1\}$ If $y_{pred} = y$ then our prediction is correct and we do nothing. Else if we predicted negative instead of positive, we multiply w_i by $(1 + \epsilon x_i)$ for all i ; if we predicted positive instead of negative then we multiply w_i by $(1 - \epsilon x_i)$ for all i.

An important point to note is that we only update our weights when we make a mistake. There are two main reasons why Winnow is particularly well suited to our task.

1. Our approach is based on augmenting the features of examples with a plethora of extra features. Winnow is known to be particularly effective in dealing with many irrelevant features. In particular, suppose the data has a linear separator of L_1 margin γ. That is, for some weights $w^* = (w_1^*, \ldots, w_d^*)$ with $\sum |w_i^*| = 1$ and some threshold t, all positive examples satisfy $w^* \cdot x \geq t$ and all negative examples satisfy $w^* \cdot x \leq t - \gamma$. Then the number of mistakes the Winnow algorithm makes is bounded by $O(\frac{1}{\gamma^2} \log d)$. For example, if the data is consistent with a majority vote of just r of the d features, where $r \ll d$, then the number of mistakes is just $O(r^2 \log d)$ [53].
2. Experience indicates that unlabeled data becomes particularly useful in large quantities. In order to deal with large quantities of data we will need fast algorithms, Winnow is a very

fast algorithm and does not require a large amount of memory.

5.3 Learning with Similarity Functions

Suppose $\mathbb{K}(x, y)$ is our similarity function and the examples have dimension k

We will create the mapping $\Phi(x) : \mathbb{R}^k \rightarrow \mathbb{R}^{k+d}$ in the following manner:

1. Draw d examples $\{x_1, x_2, \ldots, x_d\}$ uniformly at random from the dataset.
2. For each example x compute the mapping $x \rightarrow \{x, \mathbb{K}(x, x_1), \mathbb{K}(x, x_2), \ldots, \mathbb{K}(x, x_d)\}$

Although the mapping is very simple, in the next section we will see that it can be quite effective in practice.

5.3.1 Choosing a Good Similarity Function

The Naïve approach

We consider as a valid similarity function any function $\mathbb{K}(x, y)$ that takes two inputs in the appropriate domain and outputs a number between -1 and 1. This very general criteria obviously does not constrain us very much in choosing a similarity function.

But we would also intuitively like our similarity function to assign a higher similarity to pairs of examples that are more "similar." In the case where we have positive and negative examples it would seem to be a good idea if our function assigned a higher average similarity to examples that have the same label. We can formalize these intuitive ideas and obtain rigorous criteria for "good" similarity functions [2].

One natural way to construct a similarity function is by modifying an appropriate distance metric. A distance metric takes pairs of objects and assigns them a non-negative real number. If we have a distance metric $D(x, y)$ we can define a similarity function, $\mathbb{K}(x, y)$ as

$$\mathbb{K}(x, y) = \frac{1}{D(x, y) + 1}$$

Then if x and y are close according to distance metric D they will also have a high similarity score. So if we have a suitable distance function on a certain domain the similarity function constructed in this manner can be directly plugged into the Balcan-Blum algorithm.

Scaling issues

It turns out that the approach outlined previously has scaling problems, for example with the number of dimensions. If the number of dimensions is large then the similarity derived from the Euclidian distance between any two objects in a set may end up being close to zero (even if the individual features are boolean). This does not lead to a good performance.

Fortunately there is a straightforward way to fix this issue:

Ranked Similarity

We now describe an alternative way of converting a distance function to a similarity function that addresses the above problem. We first describe it in the transductive case where all data is given up front, and then in the inductive case where we need to be able to assign similarity scores to new pairs of examples as well.

Transductive Classification

1. Compute the similarity as before.

2. For each example x find the example that it is most similar to and assign it a similarity score of 1, find the next most similar example and assign it a similarity score of $(1 - \frac{2}{n-1})$, find the next one and assign it a score of $(1 - \frac{2}{n-1} \cdot 2)$ and so on until the least similar example has similarity score $(1 - \frac{2}{n-1} \cdot (n-1))$. At the end, the most similar example will have a similarity of $+1$, the least similar example will have a similarity of -1, with values spread linearly in between.

This procedure (we'll call it "ranked similarity") addresses many of the scaling issues with the naïve approach as each example will have a "full range" of similarities associated with it and experimentally it leads to much to better performance.

Inductive Classification

We can easily extend the above procedure to classifying new unseen examples by using the following similarity function:-

$$\mathbb{K}_S(x, y) = 1 - 2\text{Prob}_{z \sim S}[d(x, z) < d(x, y)]$$

where S is the set of all the labeled and unlabeled examples.

So the similarity of a new example is found by interpolating between the existing examples.

Properties of the ranked similarity

One of the interesting things about this approach is that similarity is no longer **symmetric**, as the similarity is now defined in a way analogous to nearest neighbor. In particular, you may not be the most similar example for the example that is most similar to you.

This is notable because this is a major difference with the standard definition of a kernel (as a non-symmetric function is definitely not symmetric positive definite) and provides an example

where the similarity function approach gives more flexibility than kernel methods.

Comparing Similarity Functions

One way of comparing how well a similarity function is suited to a particular dataset is by using the notion of a strongly (ϵ, γ)-good similarity function as defined by Balcan and Blum [2]. We say that $\mathbb{K}$ is a strongly (ϵ, γ)- good similarity function for a learning problem P if at least a $(1-\epsilon)$ probability mass of examples x satisfy $E_{x' \sim P}[\mathbb{K}(x', x)|l(x') = l(x)] \geq E_{x' \sim P}[\mathbb{K}(x', x)|l(x') \neq l(x)] + \gamma$ (i.e most examples are more similar to examples that have the same label).

We can easily compute the margin γ for each example in the dataset and then plot the examples by decreasing margin. If the margin is large for most examples, this is an indication that the similarity function may perform well on this particular dataset.

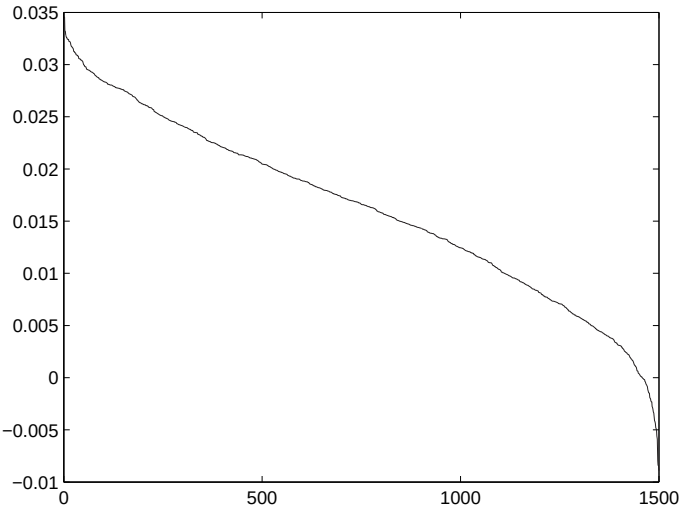

Figure 5.1: The Naïve similarity function on the Digits dataset

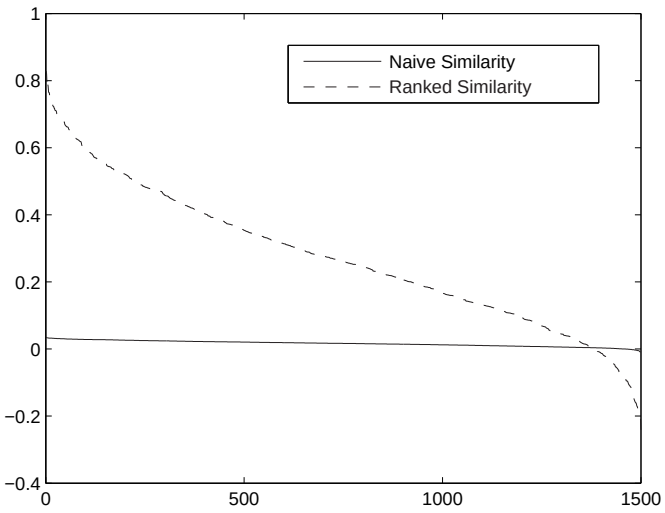

Figure 5.2: The ranked similarity and the naïve similarity plotted on the same scale

Comparing the naïve similarity function and the ranked similarity function on the Digits dataset we can see that the ranked similarity function leads to a much higher margin on most of the examples and experimentally we found that this also leads to a better performance.

5.4 Experimental Results on Synthetic Datasets

To gain a better understanding of the algorithm we first performed some experiments on synthetic datasets.

5.4.1 Synthetic Dataset:Circle

The first dataset we consider is a circle as shown in Figure 5.3 Clearly this dataset is not linearly separable. The interesting question is whether we can use our mapping to map it into a linearly separable space.

We trained it on the original features and on the induced features. This experiment had 1000 examples and we averaged over 100 runs. Error bars correspond to 1 standard deviation. The results are given in figure 5.4. The similarity function that we used in this experiment is $\mathbb{K}(x, y) = (1/(1 + ||x - y||)$

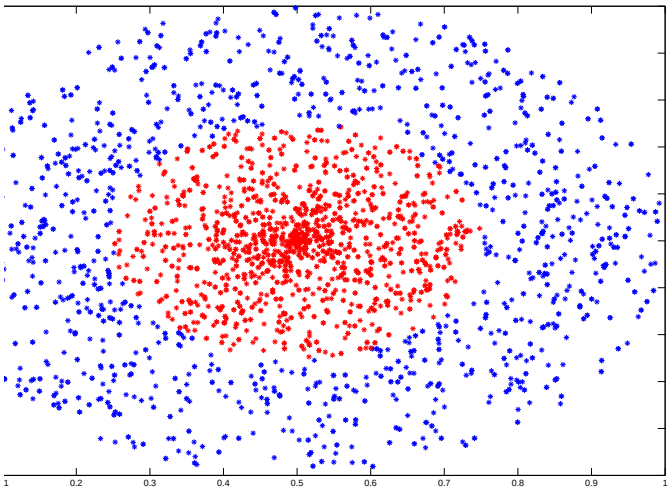

Figure 5.3: The Circle Dataset

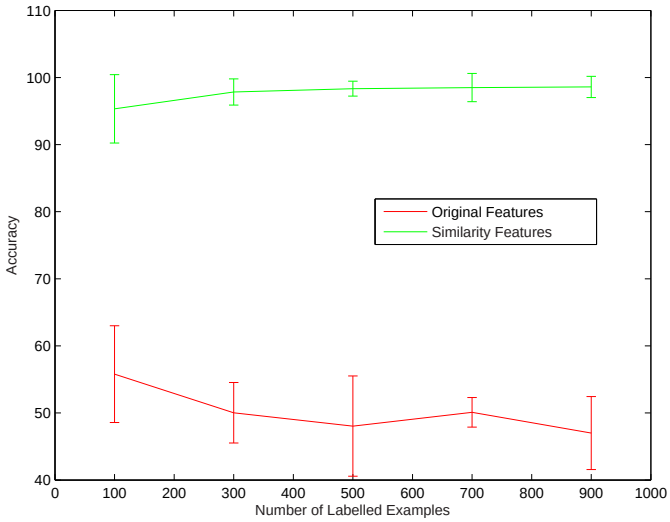

Figure 5.4: Performance on the circle dataset

5.4.2 Synthetic Dataset:Blobs and Line

We expect the original features to do well if the features are linearly separable and we expect the similarity induced features to do particularly well if the data is clustered in well-separated "blobs". One interesting question is what happens if data satisfies neither of these conditions overall, but has some portions satisfying one and some portions satisfying the other.

We generated this dataset in the following way:

1. We selected k points to be the centers of our blobs and randomly assign them labels in $-1, +1$.
2. We then repeat the following process n times:

a We flip a coin.

b If it comes up heads then we set x to random boolean vector of dimension d and $y = x_1$ (the first coordinate of x).

c If it comes up tails then we pick one of the k centers and flip r bits and set x equal to the result and set y equal to the label of the center.

The idea is that the data will be of two types, 50% is completely linearly separable in the original features and 50% is composed of several blobs. Neither one of the feature spaces by themselves should be able to represent the combination well, but the features combined should be able to work well.

As before we trained the algorithm on the original features and on the induced features. But this time we also combined the original and induced features and trained on that. This experiment had 1000 examples and we averaged over 100 runs. Error bars correspond to 1 standard deviation. The results are seen in figure 5.5. We did the experiment using both the naïve and ranked similarity functions and got similar results, and in figure 5.5 we show the results using $\mathbb{K}(x, y) = (1/(1 + ||x - y||)$.

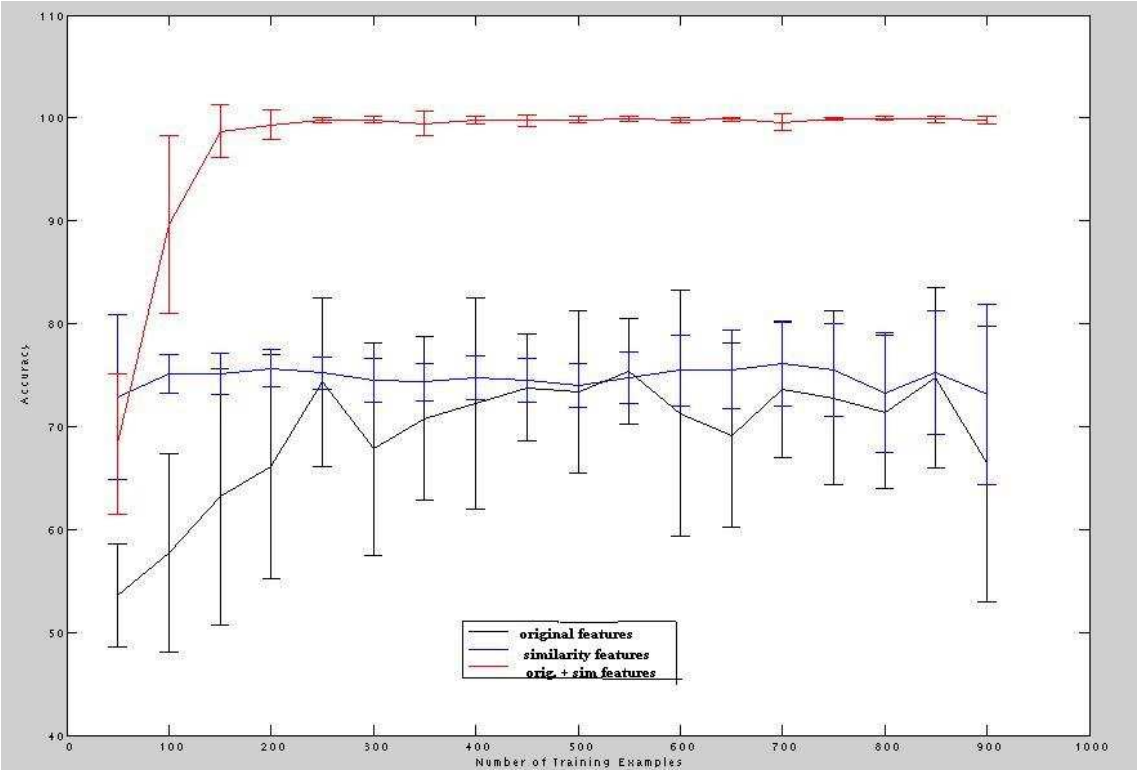

Figure 5.5: Accuracy vs training data on the Blobs and Line dataset

As expected both the original features and the similarity features get about 75% accuracy (getting almost all the examples of the appropriate type correct and about half of the examples of the other type correct) but the combined features are almost perfect in their classification accuracy. In particular this example shows that in at least some cases there may be advantages to augmenting the original features with additional features as opposed to just using the new features by themselves.

5.5 Experimental Results on Real Datasets

To test the applicability of this method we ran experiments on UCI datasets. Comparison with Winnow, SVM and NN (1 Nearest Neighbor) is included.

5.5.1 Experimental Design

For Winnow, NN, Sim and Sim+Winnow each result is the average of 10 trials. On each trial we selected 100 training examples at random and used the rest of the examples as test data. We selected 200 random examples as landmarks on each trial.

5.5.2 Winnow

We implemented Balanced Winnow with update rule ($1 \pm e^{-\epsilon X_i}$). ϵ was set to $.5$ and we ran through the data 5 times on each trial (cutting ϵ by half on each pass).

5.5.3 Boolean Features

Experience suggests that Winnow works better with boolean features, so we preprocessed all the datasets to make the features boolean. We did this by computing a median for each column and setting all features less than or equal to the median to 0 and all features greater than or equal to the median to 1.

5.5.4 Booleanize Similarity Function

We also wanted the booleanize the similarity function features. We did this by selecting for each example the 10% most similar examples and setting their similarity to 1 and setting the rest to 0.

5.5.5 SVM

For the SVM experiments we used Thorsten Joachims SVM_{light} [46] with the standard settings.

5.5.6 NN

We assign each unlabeled example the same label as that of the closest example in Euclidean distance.

In Table 5.1 below, we present the results of these algorithms on a range of UCI datasets. In this table, n is the total number of data points, d is the dimension of the space, and nl is the number of labeled examples.We highlight all performances within 5% of the best for each dataset in bold.

5.5.7 Results

In Table 5.1 below, we present the results of these algorithms on a range of UCI datasets. In this table, n is the total number of data points, d is the dimension of the space, and nl is the number of labeled examples.We highlight all performances within 5% of the best for each dataset in bold.

Dataset	n	d	nl	Winnow	SVM	NN	SIM	Winnow+SIM
Congress	435	16	100	**93.79**	**94.93**	**90.8**	**90.90**	**92.24**
Webmaster	582	1406	100	**81.97**	71.78	72.5	69.90	**81.20**
Credit	653	46	100	**78.50**	55.52	61.5	59.10	**77.36**
Wisc	683	89	100	**95.03**	**94.51**	**95.3**	**93.65**	**94.49**
Digit1	1500	241	100	73.26	88.79	**94.0**	**94.21**	**91.31**
USPS	1500	241	100	71.85	74.21	**92.0**	86.72	**88.57**

Table 5.1: Performance of similarity functions compared with standard algorithms on some real datasets

We can observe that on certain types of datasets such as the Webmaster dataset (a dataset of documents) a linear separator like Winnow performs particularly well, while standard Nearest Neighbor does not perform as well. But on other datasets such as USPS(a dataset comprised of images) Nearest Neighbor performs much better than any linear separator algorithm. The

important thing to note is that the combination of Winnow plus the similarity features always manages to perform almost as well as the best available algorithm.

5.6 Concatenating Two Datasets

In the section 5.4 we looked at some purely synthetic datasets. An interesting idea is to consider a ”hybrid” dataset obtained by combining two distinct real datasets. This models a dataset which is composed of two disjoint subsets that are part of a larger category.

We ran an experiment combining the Credit dataset and the Digit1 dataset. We combined the two datasets by padding each example with zeros so they both ended up with the same number of dimensions as seen in the table below:

Credit (653×46)	Padding (653×241)
Padding (653×46)	Digit1 (653×241)

Table 5.2: Structure of the hybrid dataset

We ran some experiments on the combined dataset using the same settings as outlined in the previous section:

Dataset	n	d	nl	Winnow	SVM	NN	SIM	Winnow+SIM
Credit+Digit1	1306	287	100	72.41	51.74	75.46	74.25	83.95

Table 5.3: Performance of similarity functions compared with standard algorithms on a hybrid dataset

5.7 Discussion

For the synthetic datasets (Circle and Blobs and Lines) the similarity features are clearly useful and have superior performance to the original features. For the UCI datasets we observe that the

combination of the similarity features with the original features does significantly better than any other approach on its own. In particular it is never significantly worse than the best algorithm on any particular dataset. We observe the same result on the ”hybrid” dataset where the combination of features does significantly better than either on its own.

5.8 Conclusion

In this report we explored techniques for learning using general similarity functions. We experimented with several ideas that have not previously appeared in the literature:-

1. Investigating the effectiveness of the Balcan-Blum approach to learning with similarity functions on real datasets.
2. Combining Graph Based and Feature Based learning Algorithms.
3. Using unlabeled data to help construct a similarity function.

From our results we can conclude that generic similarity functions do have a lot of potential for practical applications. They are more general than kernel functions and can be more easily understood. In addition by combining feature based and graph based methods we can often get the ”best of both worlds.”

5.9 Future Work

One interesting direction would be to investigate designing similarity functions for specific domains. The definition of a similarity function is so flexible that it allows great freedom to experiment and design similarity functions that are specifically suited for particular domains. This is not as easy to do for kernel functions which have stricter requirements.

Another interesting direction would be to model some realistic theoretical guarantees relating

the quality of a similarity function to the performance of the algorithm.

Bibliography

[1] Rosa I. Arriaga and Santosh Vempala. Algorithmic theories of learning. In *Foundations of Computer Science*, 1999. 5.2.3

[2] M.-F. Balcan and A. Blum. On a theory of learning with similarity functions. *ICML06, 23rd International Conference on Machine Learning*, 2006. 1.2.3, 2.3, 5, 5.1.1, 5.1.2, 5.2.4, 5.3.1, 5.3.1

[3] M.-F. Balcan, A. Blum, and S. Vempala. Kernels as features: On kernels, margins and low-dimensional mappings. *ALT04, 15th International Conference on Algorithmic Learning Theory*, pages 194—205. 1.2.1

[4] M.F. Balcan, A. Blum, and S. Vempala. Kernels as features: On kernels, margins, and low-dimensional mappings. *Machine Learning*, 65(1):79–94, 2006. 5, 5.1.1, 5.1.2, 5.2.3

[5] R. Bekkerman, A. McCallum, and G. Huang. categorization of email into folders: Benchmark experiments on enron and sri corpora,. Technical Report IR-418, University of Massachusetts,, 2004. 2.3

[6] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006. 2.1.2, 5.1.2

[7] G.M. Benedek and A. Itai. Learnability with respect to a fixed distribution. *Theoretical Computer Science*, 86:377—389, 1991. 3.4.2

[8] K. P. Bennett and A. Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems 10*, pages 368—374. MIT Press, 1998.

[9] T. De Bie and N. Cristianini. Convex methods for transduction. In *Advances in Neural Information Processing Systems 16*, pages 73—80. MIT Press, 2004. 2.1.2

[10] T. De Bie and N. Cristianini. Convex transduction with the normalized cut. Technical Report 04-128, ESAT-SISTA, 2004. 2.1.2

[11] A. Blum. Empirical support for winnow and weighted majority algorithms: results on a calendar scheduling domain. *ICML*, 1995. 2.3

[12] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the 18th International Conference on Machine Learning*, pages 19—26. Morgan Kaufmann, 2001. (document), 1.1, 1.2.1, 1.2.2, 2.1.2, 2.1.2, 3.1, 3.2, 3.5, 4.1.1, 5.1.2

[13] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In

Proceedings of the 1998 Conference on Computational Learning Theory, July 1998. 1.2.1

[14] A. Blum, J. Lafferty, M. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. *ICML04, 21st International Conference on Machine Learning*, 2004. 2.1.2, 5.1.2

[15] Avrim Blum. Notes on machine learning theory: Margin bounds and luckiness functions. http://www.cs.cmu.edu/ avrim/ML08/lect0218.txt, 2008. 5.2.1

[16] Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov random fields with efficient approximations. In *IEEE Computer Vision and Pattern Recognition Conference*, June 1998. 2.1.2

[17] U. Brefeld, T. Gaertner, T. Scheffer, and S. Wrobel. Efficient co-regularized least squares regression. *ICML06, 23rd International Conference on Machine Learning*, 2006. 1.2.2, 2.2.3, 4.1.1

[18] A. Broder, R. Krauthgamer, and M. Mitzenmacher. Improved classification via connectivity information. In *Symposium on Discrete Algorithms*, January 2000.

[19] J. I. Brown, Carl A. Hickman, Alan D. Sokal, and David G. Wagner. Chromatic roots of generalized theta graphs. *J. Combinatorial Theory, Series B*, 83:272—297, 2001. 3.3

[20] Vitor R. Carvalho and William W. Cohen. Notes on single-pass online learning. Technical Report CMU-LTI-06-002, Carnegie Mellon University, 2006. 2.3

[21] Vitor R. Carvalho and William W. Cohen. Single-pass online learning: Performance, voting schemes and online feature selection. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD 2006)*. 2.3

[22] V. Castelli and T.M. Cover. The relative value of labeled and unlabeled samples in pattern-recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory*, 42(6):2102—2117, November 1996. 1.2.1, 2.1.1

[23] C.Cortes and M.Mohri. On transductive regression. In *Advances in Neural Information Processing Systems 18*. MIT Press, 2006. 1.2.2, 2.2.1, 4.1.1

[24] S. Chakrabarty, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998.

[25] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006. URL `http://www.kyb.tuebingen.mpg.de/ssl-book`. 2, 4.1

[26] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990. 2.1.2

[27] F.G. Cozman and I. Cohen. Unlabeled data can degrade classification performance of generative classifiers. In *Proceedings of the Fifteenth Florida Artificial Intelligence Research Society Conference*, pages 327—331, 2002. 2.1.1

[28] I. Dagan, Y. Karov, and D. Roth. Mistake driven learning in text categorization. In *EMNLP*, pages 55—63, 1997. 2.3

[29] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of the johnson-lindenstrauss

lemma. Technical report, 1999. 5.2.3

[30] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1—38, 1977. 1.2.1, 2.1.1

[31] Luc Devroye, Laszlo Györfi, and Gabor Lugosi. *A Probabilistic Theory of Pattern Recognition (Stochastic Modelling and Applied Probability)*. Springer, 1997. ISBN 0387946187. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0387946187`. 1.2.1

[32] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000. 1.2.1

[33] M. Dyer, L. A. Goldberg, C. Greenhill, and M. Jerrum. On the relative complexity of approximate counting problems. In *Proceedings of APPROX'00, Lecture Notes in Computer Science 1913*, pages 108—119, 2000. 3.2

[34] Maria florina Balcan and Avrim Blum. A pac-style model for learning from labeled and unlabeled data. In *In Proceedings of the 18th Annual Conference on Computational Learning Theory (COLT*, pages 111–126. COLT, 2005.

[35] Y. Freund, Y. Mansour, and R.E. Schapire. Generalization bounds for averaged classifiers (how to be a Bayesian without believing). To appear in Annals of Statistics. Preliminary version appeared in Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics, 2001, 2003. 3.4.2

[36] Evgeniy Gabrilovich and Shaul Markovitch. Feature generation for text categorization using world knowledge. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1048—1053, Edinburgh, Scotand, August 2005. URL `http://www.cs.technion.ac.il/~gabr/papers/fg-tc-ijcai05.pdf`.

[37] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2):271—279, 1989. 3.2

[38] Steve Hanneke. An analysis of graph cut size for transductive learning. In *the 23rd International Conference on Machine Learning*, 2006. 3.4.2

[39] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2001. ISBN 0387952845. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0387952845`. 1.2.1

[40] Thomas Hofmann. Text categorization with labeled and unlabeled data: A generative model approach. In *NIPS 99 Workshop on Using Unlabeled Data for Supervised Learning*, 1999.

[41] J.J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:550—554, 1994. 1.1, 3.6.1

[42] M. Jerrum and A. Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22:1087—1116, 1993. 3.2

[43] M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method: An approach to approximate counting and integration. In D.S. Hochbaum, editor, *Approximation algorithms for* NP*-hard problems*. PWS Publishing, Boston, 1996.

[44] T. Joachims. Transductive learning via spectral graph partitioning. In *Proceedings of the* 20^{th} *International Conference on Machine Learning (ICML)*, pages 290—297, 2003. 2.1.2, 2.1.2, 3.1, 3.4.2, 3.6, 5.1.2

[45] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the* 16^{th} *International Conference on Machine Learning (ICML)*, 1999. 1.1, 2.1.2

[46] Thorsten Joachims. *Making large-Scale SVM Learning Practical*. MIT Press, 1999. 5.5.5

[47] David Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4), 1996.

[48] J. Kleinberg. Detecting a network failure. In *Proc. 41st IEEE Symposium on Foundations of Computer Science*, pages 231—239, 2000. 3.4.1

[49] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. In *40th Annual Symposium on Foundations of Computer Science*, 2000.

[50] J. Kleinberg, M. Sandler, and A. Slivkins. Network failure detection and graph connectivity. In *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 76—85, 2004. 3.4.1

[51] Paul Komarek and Andrew Moore. Making logistic regression a core data mining tool: A practical investigation of accuracy, speed, and simplicity. Technical Report CMU-RI-TR-05-27, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2005. 5.1.2

[52] John Langford and John Shawe-Taylor. PAC-bayes and margins. In *Neural Information Processing Systems*, 2002. 3.4.2

[53] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 1988. 2.3, 5.1.2, 5.2.1, 5.2.5, 1

[54] D. McAllester. PAC-bayesian stochastic model selection. *Machine Learning*, 51(1):5—21, 2003. 3.1, 3.4.2

[55] Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *COLT*, pages 154–168, 2006. 5.2.2

[56] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. 1.2.1, 5.1.2, 5.2.1

[57] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Learning to classify text from labeled and unlabeled documents. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press, 1998. 2.1.1

[58] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380—393, April 1997.

[59] Joel Ratsaby and Santosh S. Venkatesh. Learning from a mixture of labeled and unlabeled examples with parametric side information. In *Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 412—417. ACM Press, New York, NY, 1995. 1.2.1

[60] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323—2326, 2000.

[61] Sebastien Roy and Ingemar J. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *International Conference on Computer Vision (ICCV'98)*, pages 492—499, January 1998.

[62] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, 2002. 1.2.3, 5.1.1, 5.1.2, 5.2.1

[63] Bernhard Scholkopf and Mingrui Wu. Transductive classification vis local learning regularization. In *AISTATS*, 2007. 4.5.7, 5.1.2

[64] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521813972. 1.2.3, 5.1.1, 5.1.2, 5.2.1

[65] John Shawe-Taylor and Nello Cristianini. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, 1999. 1.2.3, 5.1.1, 5.1.2, 5.2.1

[66] John Shawe-taylor, Peter L. Bartlett, Robert C. Williamson, and Martin Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE transactions on Information Theory*, 44:1926–1940, 1998. 5.1.2, 5.2.1

[67] J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 731—737, 1997.

[68] V. Sindhwani, P. Niyogi, and M. Belkin. A co-regularized approach to semi-supervised learning with multiple views. *Proc. of the 22nd ICML Workshop on Learning with Multiple Views*, 2005. 1.2.1, 1.2.2, 2.2.3, 4.1.1

[69] Dan Snow, Paul Viola, and Ramin Zabih. Exact voxel occupancy with graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2000.

[70] Nathan Srebro. personal communication, 2007. 2.3

[71] Josh Tenenbaum, Vin de Silva, and John Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290, 2000.

[72] S. Thrun, T. Mitchell, and J. Cheng. The MONK's problems. a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, December 1991.

[73] UCI. Repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html, 2000.

[74] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995. 2.1.2

[75] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998. 2.1.2

[76] J.-P Vert, H. Saigo, and T. Akutsu. Local alignment kernels for biological sequences. In B. Scholkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel methods in Computational Biology*, pages 131—154. MIT Press, Boston, 2004. 1.2.3, 2.3, 5.1.1

[77] Larry Wasserman. *All of Statistics : A Concise Course in Statistical Inference (Springer Texts in Statistics)*. Springer, 2004. ISBN 0387402721. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\`

&path=ASIN/0387402721. 1.2.1, 1.2.2, 4.1.1

[78] Larry Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics)*. Springer, 2007. ISBN 0387251456. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0387251456. 1.2.1, 1.2.2, 4.1.1

[79] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15:1101—1113, 1993.

[80] Tong Zhang and Frank J. Oles. A probability analysis on the value of unlabeled data for classification problems. In *Proc. 17th International Conf. on Machine Learning*, pages 1191–1198, 2000. 1.2.1

[81] D. Zhou, O. Bousquet, T.N. Lal, J. Weston, and B. Scholkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004. 4.5.7

[82] Z.-H. Zhou and M. Li. Semi-supervised regression with co-training. *International Join Conference on Artificial Intelligence(IJCAI)*, 2005. 1.2.2, 2.2.2, 4.1.1

[83] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005. http://www.cs.wisc.edu/∼jerryzhu/pub/ssl_survey.pdf. 2

[84] X. Zhu. Semi-supervised learning with graphs. 2005. Doctoral Dissertation. 2, 5.1.2

[85] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*, pages 912—919, 2003. 1.1, 1.2.1, 1.2.2, 2.1.2, 2.1.2, 2.1.2, 3.1, 3.4.2, 3.6, 3.6.1, 3.7, 4.1.1, 4.1.2, 2, 4.5.7, 5.1.2